CROSSTALK

October 2007 **The Journal of Defense Software Engineering** Vol. 20 No. 10

SYSTEMS ENGINEERING

SOME ASSEMBLY REQUIRED

Systems Engineering

4 Using the Incremental Commitment Model to Integrate System Acquisition, Systems Engineering, and Software Engineering
This article presents the Incremental Commitment Model which emerged from a National Research Council study and related efforts; it shows promise of improving integration of hardware, software, and human factors into the systems engineering and acquisition process.
by Dr. Barry Boehm and Jo Ann Lane

10 Systems Engineering for the Global Information Grid: An Approach at the Enterprise Level
This article describes an approach at the enterprise level for systems engineering for the Global Information Grid.
by Patrick M. Kern

13 ConOps: The Cryptex to Operational System Mission Success
This article covers the Concept of Operations document – what it is and what it is not – and describes the role it played in four Air Force programs.
by Alan C. Jost

17 Software System Engineering: A Tutorial
This tutorial integrates the definitions and processes of software engineering standards into the software systems engineering process developed by the Institute of Electrical and Electronics Engineers.
by Dr. Richard Hall Thayer

Open Forum

22 Issues Using DoDAF to Engineer Fault-Tolerant Systems of Systems
This article considers aspects of the Department of Defense Architectural Framework in the systems engineering of complex systems that can be used to improve fault tolerance, describes some apparent deficiencies of the framework from a fault-tolerance perspective, and provides suggestions for improvement.
by Dr. Ronald J. Leach

28 A Framework for Evolving System of Systems Engineering
This article provides a framework for examining the differences between systems engineering and system of systems engineering, and notes that additional work is needed in the development of normative and prescriptive models.
by Dr. Ricardo Valerdi, Dr. Adam M. Ross, and Dr. Donna H. Rhodes

ON THE COVER

Cover Design by Kent Bingham

Photo Credit: Nathan Allred

Additional art services provided by Janna Jensen

Departments

3 From the Sponsor

12 Web Sites

21 Coming Events

31 BACKTALK

CROSSTALK

CO-SPONSORS:

DoD-CIO	*The Honorable John Grimes*
NAVAIR	*Jeff Schwalb*
76 SMXG	*Kevin Stamey*
309 SMXG	*Norman LeClair*
402 SMXG	*Diane Suchan*
DHS	*Joe Jarzombek*

STAFF:

MANAGING DIRECTOR	*Brent Baxter*
PUBLISHER	*Elizabeth Starrett*
MANAGING EDITOR	*Kase Johnstun*
ASSOCIATE EDITOR	*Chelene Fortier-Lozancich*
ARTICLE COORDINATOR	*Nicole Kentta*
PHONE	(801) 775-5555
E-MAIL	crosstalk.staff@hill.af.mil
CROSSTALK ONLINE	www.stsc.hill.af.mil/crosstalk

CROSSTALK, The Journal of Defense Software Engineering is co-sponsored by the Department of Defense Chief Information Office (DoD-CIO); U.S. Navy (USN); U.S. Air Force (USAF); Defense Finance and Accounting Services (DFAS); and the U.S. Department of Homeland Security (DHS). DoD-CIO co-sponsor: Assistant Secretary of Defense (Networks and Information Integration). USN co-sponsor: Naval Air Systems Command. USAF co-sponsors: Oklahoma City-Air Logistics Center (ALC) 76 Software Maintenance Group (SMXG); Ogden-ALC 309 SMXG; and Warner Robins-ALC 402 SMXG. DHS co-sponsor: National Cyber Security Division of the Office of Infrastructure Protection.

The USAF Software Technology Support Center (STSC) is the publisher of CROSSTALK, providing both editorial oversight and technical review of the journal. CROSSTALK's mission is to encourage the engineering development of software to improve the reliability, sustainability, and responsiveness of our warfighting capability.

S T S C

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail us or use the form on p. 27.

517 SMXS/MXDEA
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the CROSSTALK editorial board prior to publication. Please follow the Author Guidelines, available at <www.stsc.hill.af.mil/crosstalk/xtlkguid.pdf>. CROSSTALK does not pay for submissions. Articles published in CROSSTALK remain the property of the authors and may be submitted to other publications.

Reprints: Permission to reprint or post articles must be requested from the author or the copyright holder and coordinated with CROSSTALK.

Trademarks and Endorsements: This Department of Defense (DoD) journal is an authorized publication for members of the DoD. Contents of CROSSTALK are not necessarily the official views of, or endorsed by, the U.S. government, the DoD, the co-sponsors, or the STSC. All product names referenced in this issue are trademarks of their companies.

CrossTalk Online Services: See <www.stsc.hill.af.mil/crosstalk>, call (801) 777-0857 or e-mail <stsc.webmaster@hill.af.mil>.

Back Issues Available: Please phone or e-mail us to see if back issues are available free of charge.

Revitalization of Systems Engineering Within the Department of Defense and the Expanding Role of Software

As the Department of Defense (DoD) moves into more complex systems of systems and family of systems with the accompanying challenges of architectures and product lines, software will further dominate development and deployment. Future systems engineers will be required to have more knowledge of software in addition to hardware. This will be a challenging transition for many, yet it is a transition that must be made if we are to be successful in the revitalization of systems engineering for the DoD. The importance of the role of software development in systems engineering processes cannot be overstated.

One of the great misunderstandings of systems engineering and associated processes is the role of software. I believe we are in the midst of a paradigm shift within the DoD. Although developing and building hardware is difficult, we know how to do it. Conversely, few senior leaders or program managers appreciate what is involved in the development and deployment of software. Additionally, not many acquisition professionals recognize the impact of system or requirements changes on software development. These gaps in understanding usually result in software developers being held accountable for schedule slips and budget overruns, which increase alienation of the software community from hardware developers and the systems engineering process.

Across the DoD, significant effort is being invested in the revitalization of the technical work force, especially systems engineering. The focus is on formal education, job training of existing technical staffs, and recruitment of both new and experienced engineers to fill vacancies. While the emphasis is on placing skilled personnel in support of acquisition and in-service engineering programs, there is also the realization that to be successful in recruiting new university graduates, the DoD must offer interesting and exciting hands-on work. Developing and maintaining in-house technical tasks is a priority.

We also need to educate program managers and senior leadership on the software development process, the unique skills of our software work force, and the role of software development in systems engineering processes. In this issue of CrossTalk, readers will find information that addresses the implications of software's influence on systems engineering now and into the future.

The basics are covered in *Software System Engineering: A Tutorial* by Dr. Richard Hall Thayer where he lays out how software ties the system together. *A Framework for Evolving System of Systems Engineering* by Dr. Ricardo Valerdi, Dr. Adam M. Ross, and Dr. Donna H. Rhodes provides a framework for examining differences between systems engineering and system of systems engineering. Other articles take the reader forward into the software systems engineering world. The article *Using the Incremental Commitment Model to Integrate System Acquisition, Systems Engineering, and Software Engineering* by Dr. Barry Boehm and Jo Ann Lane leads the reader down the path to improved integration of hardware, software, and human factors. Finally, *Systems Engineering for the Global Grid: An Approach at the Enterprise Level* by Patrick M. Kern, *ConOps: The Cryptex to Operational System Mission Success* by Alan C. Jost, and *Issues Using DoDAF to Engineer Fault-Tolerant Systems of Systems* by Dr. Ronald J. Leach take the reader into a much more complex world as we head into the (as yet) uncharted and increasingly complex software systems of the future.

We hold the future success of software systems engineering in our hands. Enjoy these insightful CrossTalk articles that delve into the world of software systems engineering. I trust you will take insights offered in this issue of CrossTalk and infuse them into your efforts to develop software and systems engineering of the future.

Dr. John W. Fischer
Naval Air Systems Command

Using the Incremental Commitment Model to Integrate System Acquisition, Systems Engineering, and Software Engineering

Dr. Barry Boehm and Jo Ann Lane
University of Southern California Center for Systems and Software Engineering

One of the top recommendations to emerge from the October 2006 Deputy Under Secretary of Defense (DUSD) Acquisition, Technology, and Logistics (ATL) Defense Software Strategy Summit was to find ways of better integrating software engineering into the systems engineering and acquisition process. Concurrently, the National Research Council (NRC) study was addressing the problem of better integrating human factors into the systems engineering and acquisition process. This article presents a model that emerged from these and related efforts that shows promise of improving integrations. This model, called the Incremental Commitment Model (ICM), organizes systems engineering and acquisition processes in ways that better accommodate the different strengths and difficulties of hardware, software, and human factors of engineering approaches. It also provides points at which they can synchronize and stabilize, and at which their risks of going forward can be better assessed and fitted into a risk-driven stakeholder resource commitment process.

Many projects have difficulties in integrating their hardware, software, and human factor aspects. Basically, this is due to differences between these aspects with respect to their underlying economics, evolution patterns, product subsetability (ability to deliver usable, partial, initial, operational capabilities), user tailorability, adaptability, underlying science, and testing considerations.

This article begins by summarizing trends that have caused difficulties for current systems engineering and acquisition processes and underlying principles that better address these trends. It then presents several complementary views of the ICM, discusses their implications with respect to acquisition and engineering practices and personnel career paths, and it assesses project performance with respect to use of the principles. The principles can also be used to avoid the negative effects of common misinterpretations of other current models such as the V, spiral, and Rational Unified Process (RUP) models.

Summary of Difficulties and Some of Their Causes

Current systems engineering and acquisition practices (and the associated program management personnel) still rely heavily on their historical hardware engineering and acquisition legacy. An emphasis on reducing hardware development and manufacturing costs often leads to selection of components with incompatible software infrastructures and human interfaces, leading to much higher development, operations, and maintenance costs, as well as associated system underperformance in the software and human engineering areas. A hardware-oriented, fixed-price, build-to-specification contract may deliver a hardware initial operational capability (IOC) within its development budget, but may elect not to architect or upgrade the software in order to avoid excessive software maintenance or human operational costs.

The relative difficulty of modifying hardware installed in many places, as compared to electronic modification of software or modification of human operational procedures, may lead to added software costs for hardware shortfall workarounds or to fitting the people to the product rather than fitting the product to the people. And the limited subsetability of hardware systems (e.g., aircraft without landing gear or complete flight controls) as compared to partial software or human interface features often leads to incompatibilities between single-increment hardware acquisition practices and multiple-increment software and human interface practices on the same project.

If these hardware-software-human integration problems are difficult today, they will present formidable problems for the future if not adequately addressed. Some trends that will exacerbate such integration problems are the following:

1. **Complex, multi-owner Systems of Systems (SoS).** Current collections of incompatible, separately developed systems cause numerous operational deficiencies such as unacceptable delays in service, uncoordinated and conflicting plans, ineffective or dangerous decisions, and inability to cope with fast-moving events. Multiple owners of key interdependent systems make integration of SoS a major challenge, but the current alternative of just trying to mash them together will only become worse in the future [1].
2. **Emergent requirements.** The most appropriate user interfaces and collaboration modes for a complex human-intensive system are not specifiable in advance, but emerge with system prototyping and usage. Forcing them to be prematurely and precisely specified generally leads to poor business or mission performance and expensive, late rework and delays [2].
3. **Rapid change.** Specifying current point-in-time snapshot requirements on a cost-competitive contract generally leads to a big design up front and a point-solution architecture that is hard to adapt to new developments. Each of the many subsequent changes then leads to considerable nonproductive work in redeveloping documents and software (or worse, hardware), and in renegotiating contracts [3].
4. **Reused components.** Building all of one's own components from scratch will be economically infeasible for complex systems. However, reuse-based development has major bottom-up development implications, and is incompatible with pure, top-down, requirements-first approaches. Prematurely specifying requirements (e.g., hasty specification of a one-second response time requirement when later prototyping shows that four seconds would be acceptable) that disqualify otherwise most cost-effective reusable components often leads to overly expensive, late, and unsatisfactory systems [4].
5. **High assurance of qualities.** Future systems will need higher assurance levels of such qualities as safety, security, reliability/availability/maintainability, perfor-

mance, adaptability, interoperability, usability, and scalability. Just assuring one of these qualities for a complex SoS will be difficult. Given the need to *satisfice – not everybody gets everything they want, but everybody gets something they are satisfied with* – among multiple system owners with different quality priorities, their complex sources of conflict and trade-off relationships will make multi-attribute satisficing even more challenging [5].

Such concerns led to one of the top recommendations from the October 2006 DUSD ATL Defense Software Strategy Summit. This recommendation was to find ways of better integrating software engineering into the systems engineering and acquisition process [6]. Concurrently, a NRC study was addressing the problem of better integrating human factors into the systems engineering and acquisition process [7].

Several analyses were performed to determine the kind of process that would satisfactorily address these challenges. As part of the NRC study, the strengths and difficulties of current process models were analyzed. Each had strengths but needed further refinements to address all of the previous five challenges. The most important conclusion, though, was that there were key process principles that address the challenges, and that forms of the models were less important than their ability to adopt the principles. These key principles are the following:

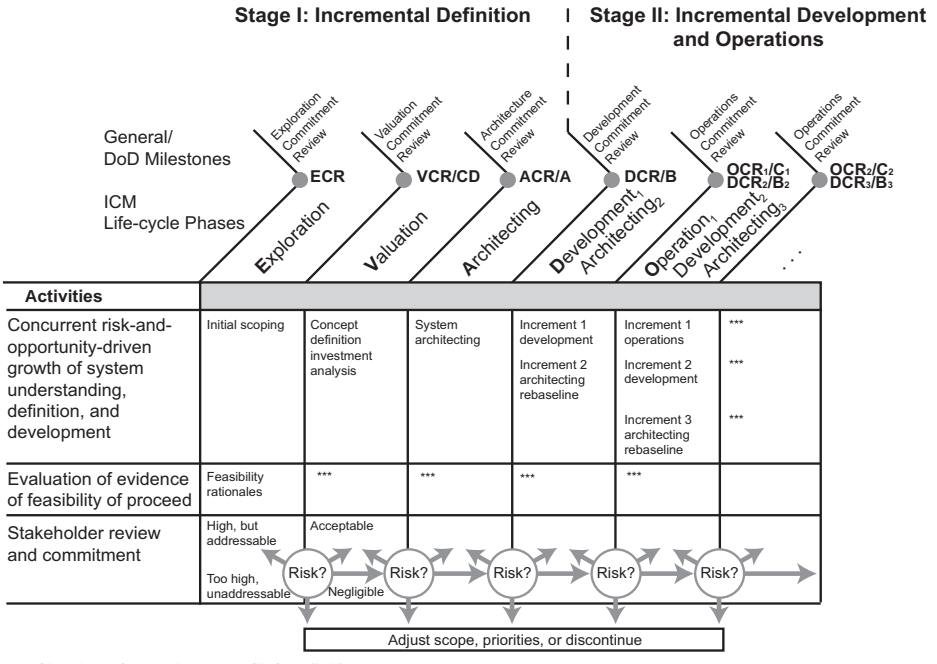
1. **Stakeholder satisficing.** If a system development process presents a success-critical operational or development stakeholder with the prospect of an unsatisfactory outcome, the stakeholder will generally refuse to cooperate, resulting in an unsuccessful system. Stakeholder satisficing includes identifying the success-critical stakeholders and their value propositions; negotiating a mutually satisfactory set of system requirements, solutions, and plans; and managing proposed changes to preserve a mutually satisfactory outcome.
2. **Incremental and evolutionary growth of system definition and stakeholder commitment.** This characteristic captures the often incremental discovery of emergent requirements and solutions via methods such as prototyping, operational exercises, and use of early system capabilities. Requirements and commitment cannot be monolithic or fully pre-specifiable for complex, human-intensive systems; increasingly detailed understanding, trust, definition and commitment is achieved through an evolutionary process.
3. **Iterative system development and definition.** The incremental and evolutionary approaches lead to cyclic refinements of requirements, solutions, and development plans. Such iteration helps projects to learn early and efficiently about operational and quality requirements and priorities.
4. **Concurrent system definition and development.** Initially, this includes concurrent engineering of requirements and solutions, and integrated product and process definition. In later increments, change-driven rework and rebaselining of next-increment requirements, solutions, and plans occurs concurrently with stabilized development of the current system increment. This allows early fielding of core capabilities, continual adaptation to change, and timely growth of complex systems without waiting for every requirement and subsystem to be defined.
5. **Risk management – risk-driven anchor-point milestones.** The key to synchronizing and stabilizing all of this concurrent activity is a set of risk-driven anchor point milestones. At these milestones, the business, technical, and operational feasibility of the growing package of specifications and plans is evaluated by independent experts. Shortfalls in feasibility evidence are treated as risks and addressed by risk management plans. If the system's success-critical stakeholders find the risks acceptable and the risk management plans sound, the project will proceed to the next phase. If not, the project can extend its current phase, de-scope its objectives, or avoid low-return resource commitments by terminating the project. If the risks of proceeding straight into development are negligible, the project can skip one or more of the early phases.

Overview of the ICM

The ICM builds on the strengths of current process models: early verification and validation (V&V) concepts in the V-model, concurrency concepts in the Concurrent Engineering model, lighter-weight concepts in the Agile and Lean models, risk-driven concepts in the spiral model, the phases and anchor points in the RUP [8, 9, 10], and recent extensions of the spiral model to address SoS acquisition [11].

An overview of the ICM life-cycle process is shown in Figure 1. In comparison to the software-intensive RUP, the ICM also addresses hardware and human factor integration. It extends the RUP phases to cover the full system life cycle: An Exploration phase precedes the RUP Inception phase, which is refocused on valuation and investment analysis. The RUP Elaboration phase is refocused on architecting (a term based on describing concurrent development of requirements, architecture, and plans [12]), which adds feasibility evidence; the RUP Construction and Transition phases are combined into the Development phase; and an additional Operation phase combines operations, production, maintenance, and phase-out. Also, the names of the milestones are changed to emphasize that their objectives are to ensure stakeholder commitment to proceed to the next level of resource expenditure based on a thorough feasibility and risk analysis, and not just on

Figure 1: *Overview of the Incremental Commitment Life-Cycle Process*



Note: CD, A, B, and C are the Department of Defense (DoD) acquisition milestones.

the existence of a set of system objectives and a set of architecture diagrams. Thus, the RUP Life-Cycle Objectives (LCO) milestone is called the Architecture Commitment Review (ACR) in the ICM, and the RUP Life-Cycle Architecture (LCA) milestone is called the Development Commitment Review (DCR).

In comparison to the sequential waterfall [13] and V-model [14], the ICM explicitly does the following:

- Emphasizes concurrent engineering of requirements and solutions.
- Establishes feasibility rationales as pass/ fail milestone criteria.
- Enables risk-driven avoidance of unnecessary documents, phases, and reviews.
- Provides support for a stabilized current-increment development concurrently with a separate change processing and rebaselining activity to prepare for appropriate and stabilized development of the next increment.

These aspects can be integrated into a waterfall or V-model, enabling projects required to use such models to cope more effectively with systems of the future.

The overall life-cycle process divides naturally into two major stages. Stage I, Incremental Definition, covers the up-front growth in system understanding, definition, feasibility assurance, and stakeholder commitment, leading to a larger Stage II commitment to a feasible set of specifications and plans for Incremental Development and Operations.

Stage I: The duration of Stage I can be anywhere from one week to five years. The duration depends on such factors as the number, capability, and compatibility of the proposed system's components and stakeholders. A small, well-jelled agile-method, developer-customer team operating on a mature infrastructure can form and begin incremental development using Scrum, eXtreme Programming, Crystal, or other agile methods in a week. An ultra-large, unprecedented, multi-mission, multi-owner SoS project may take up to five years to progress from a system vision through sorting out needs, opportunities, and organizational roles; maturing key technologies; reconciling infrastructure incompatibilities; and evolving a feasibility-validated set of specifications and plans for Stage II. These specifications and plans would be at the build-to level for the initial increment, but only elaborated into detail for the later increments and the overall system where there were high-risk elements to resolve.

As shown in Figure 1, each project's activity trajectory will be determined by the risk assessments and stakeholder commitment decisions at its anchor point milestone reviews. The small agile project will follow the negligible-risk arrows at the bottom of Figure 1 to skip the Valuation and Architecting phases and begin Stage II after a short exploratory phase confirms that the risks of doing so are indeed negligible. The ultra-large project could, for example, fund eight small competitive concept-definition and validation contracts in the Exploratory phase, four larger follow-on Valuation contracts, and two considerably larger Architecting contracts, choosing at each anchor point milestone the best-qualified teams to proceed, based on the feasibility and risk evaluations performed at each anchor point milestone review. Or, in some cases, the reviews might indicate that certain essential technologies or infrastructure incompatibilities need more work before proceeding into the next phase.

Stage II: For Stage II, Incremental Development and Operations, a key decision that is made at the Development Commitment review is the length of the increments to be used in the system's development and evolution. A small agile project can use two- to four-week increments. However, an ultra-large SoS project with a couple dozen system suppliers, each with a half-dozen subcontractors, would need increments of up to two years to develop and integrate an increment of operational capability, although with several internal integration sub-increments. Some of the non-subsetable hardware systems would take even longer to develop their initial increments and would be scheduled to synchronize their deliveries with later increments.

The features in each Stage II increment would be prioritized and the increment architected to enable what has variously been called timeboxing, time-certain development, or schedule-as independent variable, in which borderline-priority features are added or dropped to keep the increment on schedule. It would also be architected to accommodate foreseeable changes, such as user interfaces or transaction formats. For highly mission-critical systems, it would include a continuous V&V team analyzing, reviewing, and testing the evolving product to minimize delayed-defect-finding rework.

While the stabilized development team is building the current increment and accommodating foreseeable changes, a separate system engineering team is dealing with sources of unforeseeable change and rebaselining the later increments' specifications and plans. Such changes can include new commercial off-the-shelf (COTS) releases, previous-increment usage feedback, current-increment deferrals to the next increment, new technology opportunities, or changes in mission priorities. Having the development team try to accommodate these changes does not work, as it destabilizes their schedules and carefully worked-out interface specifications. At the end of each increment, the system engineering team also produces for expert review the feasibility evidence necessary to ensure low-risk, stabilized development of the next increment by the build-to-spec team.

ICM Commitment Milestones: These milestones correspond fairly closely with the DoD acquisition milestones CD, A, B, and C as defined in DoD Instruction 5000.2 [15]. The ICM commitment milestones occur at similar points in the acquisition life cycle but provide additional guidance and rigor for evaluating feasibility prior to commitment for the next stage. For example, the ICM DCR milestone commitment to proceed into Development based on the validated LCA package (an Operations Concept description, Requirements description, Architecture description, Life-Cycle plan, working prototypes for high-risk elements, and a Feasibility Rationale providing evidence of their compatibility and feasibility), corresponds fairly closely with DoD's Milestone B commitment to proceed into the System Development and Demonstration phase.

ICM Metaphor: A simple metaphor to help understand the ICM is to compare ICM to poker games such as Texas Hold'em versus the single-commitment gambling games such as Roulette. Many system development contracts operate like Roulette, in which a full set of requirements is specified up front, the full set of resources is committed to an essentially fixed-price contract, and one waits to see if the bet was a good one or not. With the ICM, one places a smaller bet to see whether the prospects of a win are good or not and decides to increase the bet based on better information about the prospects of success.

What Is Being Concurrently Engineered in the ICM?

Having addressed the stages, phases, and milestones in the ICM, let us now look at the activities. The top row of Figure 1 indicates that a number of system aspects are being concurrently engineered at an increasing level of understanding, definition, and development. The most significant of these aspects are shown in Figure

2, an extension of a similar view of concurrently engineered software projects developed as part of the RUP [9].

As with the RUP version, it should be emphasized that the magnitude and shape of the levels of effort will be risk-driven and likely to vary from project to project. In particular, they are likely to have mini risk/opportunity-driven peaks and valleys, rather than the smooth curves shown for simplicity in Figure 2. The main intent of this view is to emphasize the necessary concurrency of the primary success-critical activities shown as rows in Figure 2. Thus, in interpreting the Exploration column, although system scoping is the primary objective of the Exploration phase, doing it well involves a considerable amount of activity in understanding needs, envisioning opportunities, identifying and reconciling stakeholder goals and objectives, architecting solutions, life-cycle planning, evaluating alternatives, and negotiating stakeholder commitments.

For example, if one were exploring the initial scoping of an SoS for a metropolitan area's disaster relief, one would not just interview a number of stakeholders and compile a list of their expressed mission needs. One would also *envision and explore opportunities* for reusing (parts of) other metropolitan area disaster relief systems; for obtaining development funds from federal agencies; and for applying maturing virtual collaboration technologies. In the area of understanding needs, one would concurrently assess the capability and compatibility of existing disaster relief systems in the metropolitan area to determine which would need the most work to re-engineer into an SoS. One would also assess the scope of authority and responsibility of each existing system to determine whether the best approach would be a truly integrated and centrally managed SoS or a best-effort interoperable set of systems. And one would explore alternative *architectural concepts* for developing and evolving the system; *evaluate* their relative feasibility, benefits, and risks for stakeholders to review; and *negotiate commitments* of further resources to proceed into a Valuation phase.

How Is All This Concurrent Engineering Synchronized and Stabilized?

Figure 2 indicates that a great deal of concurrent activity occurs within and across the various ICM phases. To make this concurrency work, the anchor-point milestone reviews are the mechanism by which the many concurrent activities are synchronized, stabilized, and risk-assessed at the end of each phase. Each of these anchor-point milestone reviews, labeled at the top of Figure 2, is focused on developer-produced *evidence* – instead of PowerPoint charts and Unified Modeling Language diagrams – to help the key stakeholders determine the next level of commitment. For the Exploration Commitment Review (ECR), the focus is on a review of an Exploration phase plan with the proposed scope, schedule, deliverables, and required resource commitment by a key subset of stakeholders. The plan content is risk-driven, and could therefore be put on a single page for a small and non-controversial Exploration phase since there is minimal risk at this point – a much riskier Exploration phase would require a more detailed plan outlining how the risks will be re-evaluated and managed going forward. For the Valuation Commitment Review (VCR), the risk-driven focus is similar – the content includes the Exploration phase results and a Valuation phase plan, and a review by all of the stakeholders involved in the Valuation phase. The ACR and the DCR reviews are based on the highly successful AT&T Architecture Review Board procedures described in [16]. For the ACR, only high-risk aspects of the Operational Concept, Requirements, Architecture, and Plans are elaborated in detail. And it is sufficient to provide evidence that at least one combination of those artifacts satisfies the Feasibility Rationale criteria shown in Table 1 (similar to the RUP LCO milestone), as compared to demonstrating this at the DCR for a particular choice of arti-

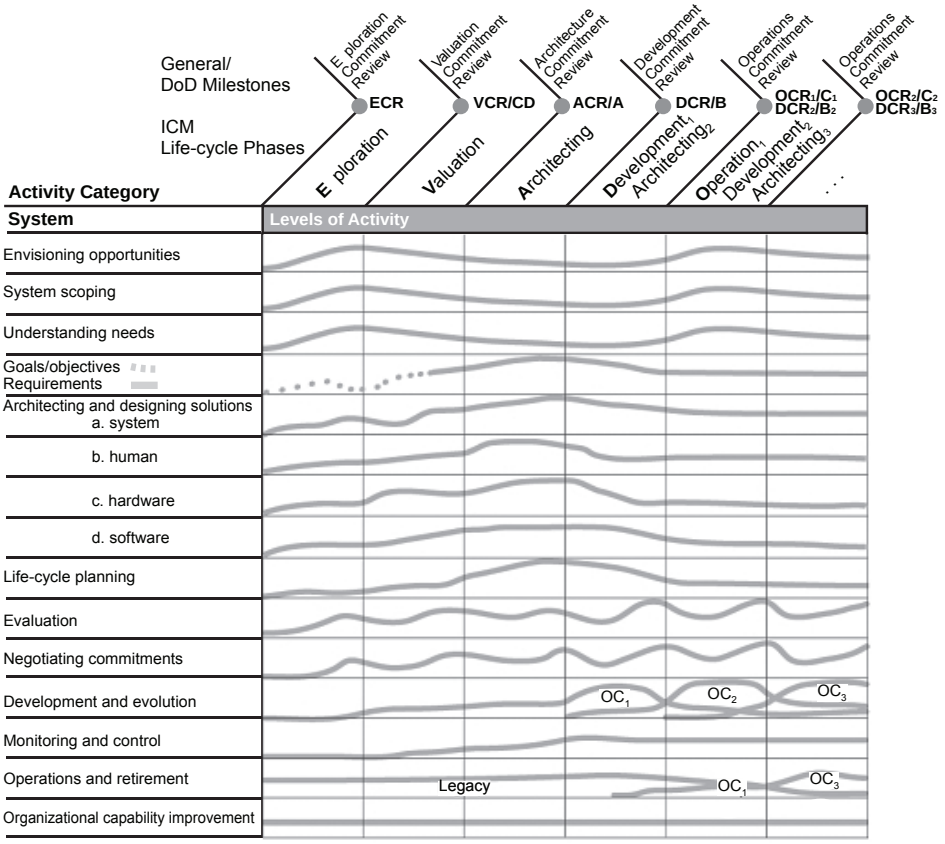


Figure 2: *ICM Activity Categories and Level of Effort*

Table 1: *Pass/Fail Feasibility Rationale Overview*

Pass/Fail Feasibility Rationales
Evidence *provided by developer* and *validated by independent experts* that if the system is built to the specified architecture, it will do the following: • Satisfy the requirements: capability, interfaces, level of service, and evolution. • Support the operational concept. • Be buildable within the budgets and schedules in the plan. • Generate a viable return on investment. • Generate satisfactory outcomes for all of the success-critical stakeholders. Resolves or covers all major risks by risk management plans. Serve as basis for stakeholders' commitment to proceed.

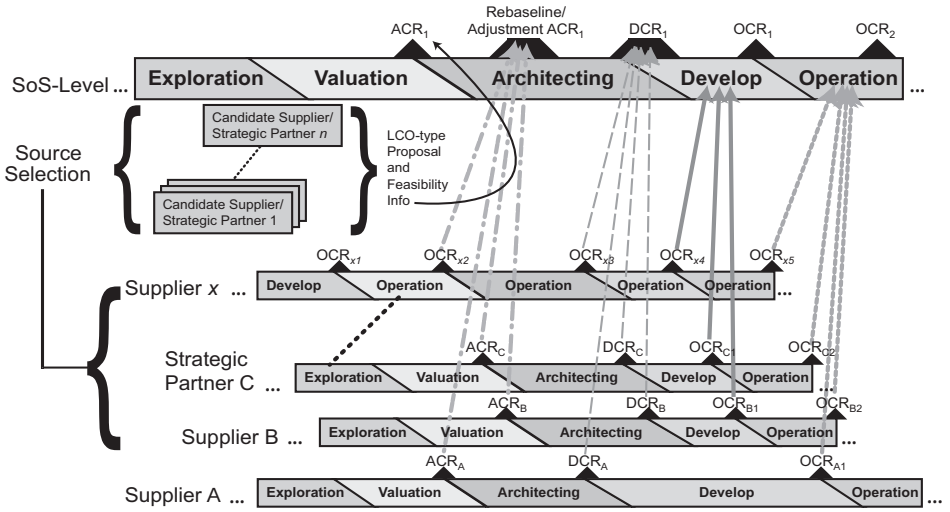


Figure 3: *Combining SoS Engineering and Component Supplier Processes Using ICM Anchor Point Reviews*

facts to be used for development.

The Operations Commitment Review (OCR) is different in that it addresses the often much higher operational risks of fielding an inadequate system. In general, stakeholders will experience a factor of two to 10 increase in commitment level in going through the sequence of ECR to DCR milestones, but the increase in going from DCR to OCR can be much higher. These commitment levels are based on typical cost profiles across the various stages of the acquisition life cycle. The OCR focuses on evidence of the adequacy of plans and preparations with respect to doctrine, organization, training, material, leadership, personnel, and facilities along with plans, budgets, and schedules for production, fielding, and operations.

Taking this to the next level for SoS development, Figure 3 shows how these anchor point milestone reviews can be used to synchronize, stabilize, and manage risks across multiple supplier/vendor/ strategic partner activities.

The major SoS-level milestones are compatible with those of Figure 1, but the realities of many SoS involve some reinterpretation of their nature. Many SoS will need to include COTS, legacy, or separately managed systems that are defined and incrementally released on different schedules than the SoS in Figure 3. The case that is shown in Figure 3 is one in which for reasons of training, provisioning, or operational stability, the main upgrades are batched into major SoS operational releases. Other SoS cases may require a more continual stream of upgrades such as security patches or electronic warfare countermeasures, in which parts of the SoS are more continuously evolving versus incrementally evolving.

Since not all the source-selected component systems are defined on the same schedule, there will be delays in reconciling them into a common system architectural framework, requiring an additional SoS rebaseline/adjustment ACR after the original SoS ACR used to drive source selection. The selected suppliers and partners will also need to participate in negotiating the particular SoS build-to architecture that will be baselined at the SoS DCR, and their incremental delivery schedules will not all be compatible with the incremental delivery schedule of the SoS in Figure 3.

Some of these suppliers, such as supplier X, will already be operating and will be feeding incremental upgrade information into the SoS process during its definition and development stages. The SoS system engineers will try to predefine and anticipate these upgrades as much as possible, but will have to adapt to changes in supplier X's capabilities or interfaces. If these are well-anticipated, the SoS development team in Figure 1 will accommodate them. If they are complex and unanticipated, threatening destabilization of the build-to-spec team, the SoS architecting-rebaselining team will engineer an interim solution if necessary to keep the operational capability running. If a more comprehensive solution is needed for the longer term, the team will rebaseline the SoS architecture of the next increment to accommodate supplier X's new capability. Changes from strategic partner C and supplier B would be handled similarly.

In some additional cases, particularly for supplier A who may be developing a longer-duration, non-subsetable, hardware-intensive IOC, the SoS management will schedule supplier A's IOC to synchronize with a later SoS increment (OCR_2 in Figure 3).

Chapter 2 of [7] provides several additional views of the ICM, including a more detailed version of Figure 1, some examples of how different risk patterns create different process sequences, a spiral view of the ICM phases and commitment milestones, and an illustration of how the concurrent increment development, increment V&V, and next-increment rebaselining activities address the need to simultaneously achieve high assurance and adaptiveness to rapid change.

Table 2: *Number of Top 5 Projects Explicitly Using ICM Principles*

Year	Concurrent Engineering	Risk-Driven	Evolutionary Growth
2002	4	3	3
2003	5	4	3
2004	3	3	4
2005	4	4	5
Total (of 20)	**16**	**14**	**15**

Project Experience With ICM Principles

The ICM uses the critical success factor principles to extend several current spiral-related processes such as RUP, WinWin Spiral Model, and Lean Development in ways that more explicitly incorporate human-system factors into the system life-cycle process. Some case studies of applying the ICM to a remotely piloted vehicle, port security, and commercial medical infusion pump development are in Chapter 5 of [7]. Another good source of successful projects that have applied the critical success factor principles is the 2002-2005 series of Top 5 software-intensive systems projects published in CrossTalk [17].

The Top 5 Quality Software Projects were chosen annually by panels of leading experts as role models of best practices and successful outcomes. Table 2 summarizes each year's record with respect to usage of four of the five principles: concurrent engineering, risk-driven activities, and evolutionary and iterative system growth (most of the projects were not specific about stakeholder satisficing). Of the 20 Top 5 projects in 2002 through 2005, 16 explicitly used concurrent engineering, 14 explicitly used risk-driven development, and 15 explicitly used evolutionary and iterative system growth, while additional projects gave indications of their partial use. Evidence of successful results of stakeholder-satisficing can be found in the annual series of University of Southern California (USC) e-services projects using the WinWin Spiral Model as described in [18]. Since 1998, more than 50 user-intensive e-services applications have used the WinWin Spiral Model to achieve a 92 percent success rate of on-time delivery of stakeholder-satisfactory systems.

Conclusion

Future transformational, net-centric systems will have many usage uncertainties and emergent characteristics. Their hardware, software, and human factors will need to be concurrently engineered, risk-managed, and evolutionarily developed to converge on cost-effective system operations. They will need to be both highly dependable and rapidly adaptable to frequent changes. The ICM described in this article builds on experience-based critical success factor principles (stakeholder satisficing, incremental definition, iterative evolutionary growth, concurrent engineering, risk management) and the strengths of existing V, concurrent engineering, spiral, agile, and lean process models to provide a framework for concurrently engineering system-specific critical factors into the systems engineering and systems development processes.

Unfortunately, the current path of least resistance for a government program manager is to follow a set of legacy regulations, specifications, and standards that select, contract with, and reward developers for doing almost the exact opposite. Most of these legacy instruments emphasize sequential versus concurrent engineering; risk-insensitive versus risk-driven processes; early definition of poorly understood requirements versus better understanding of needs and opportunities; and slow, unscalable contractual mechanisms for adapting to rapid change.

This article has provided a mapping of the ICM milestones to the current DoD 5000.2 acquisition milestones that shows that they can be quite compatible. It also shows how projects could be organized into stabilized build-to-specification increments that fit current legacy acquisition instruments, along with concurrent agile change-adaptation and V&V functions that need to use alternative contracting methods. Addressing changes of this nature will be important, particularly if organizations are to realize the large potential value offered by investments in future net-centric SoS. However, as military planners have had to recognize that there is no longer a forward edge of the battle area marking the boundary between blue and red forces, acquisition and financial planners will need to recognize that there is no longer a cutover-type milestone marking the boundary between research and development and operations and maintenance. Instead, future life cycles will need to adapt to continuous incremental and evolutionary acquisition and development.◆

References

1. Krygiel, A. "Behind the Wizard's Curtain." Command and Control Research Program Publication Series, 1999.
2. Highsmith, J. Adaptive Software Development. New York: Dorset House, 2000.
3. Beck, K. Extreme Programming Explained. Reading, MA: Addison Wesley, 1999.
4. Boehm, B. "Unifying Software Engineering and Systems Engineering." Computer Mar. 2000: 114-116.
5. Clements, P. et al. Documenting Software Architectures. Addison Wesley, 2002.
6. Baldwin, K. "DoD Software Engineering and System Assurance: New Organization, New Vision." Washington: DUSD 2007 <http://csse.usc.edu/events/2007/ARR/presentations/Baldwin.ppt>.
7. Pew, R.W., and A.S. Mavor. Human-System Integration in the System Development Process: A New Look. National Academy Press, 2007.
8. Royce, W.E. Software Project Management. Addison Wesley, 1998.
9. Kruchten, P. The Rational Unified Process. Addison Wesley, 1999.
10. Boehm, B. "Anchoring the Software Process." Software July 1996: 73-82.
11. Boehm, B., and J. Lane. "21st Century Processes for Acquiring 21st Century Systems of Systems." CROSSTALK May 2006 <www.stsc.hill.af.mil/crosstalk>.
12. Rechtin, E. Systems Architecting. Prentice Hall, 1991.
13. Royce, W.W. "Managing the Development of Large Software Systems: Concepts and Techniques." 1970 WESCON Technical Papers. v.~14. Western Electronic Show and Convention, Los Angeles, CA., Aug. 25-28, 1970.
14. Patterson, F. "System Engineering Life Cycles: Life Cycles for Research, Development, Test, and Evaluation; Acquisition; and Planning and Marketing." Handbook of Systems Engineering and Management. Ed. A. Sage and Rouse. W. Wiley, 1999: 59-111.
15. DoD. "Operation of the Defense Acquisition System." DoD Instruction 5000.2: DoD, 2003.
16. Marenzano, J., et al. "Architecture Reviews: Practice and Experience." IEEE Software Mar./Apr. 2005: 34-43.
17. CROSSTALK. "Top Five Quality Software Projects." Jan. 2002, July 2003, July 2004, Sept. 2005.
18. Boehm, B., et al. "Using the WinWin Spiral Model: A Case Study." Computer July 1998: 33-44.

About the Authors

Barry Boehm, Ph.D., is the TRW professor of software engineering and director of the Center for Systems and Software Engineering at the University of Southern California (USC). He was previously in technical and management positions at General Dynamics, Rand Corporation, TRW, and the Defense Advanced Research Projects Agency, where he managed the acquisition of more than $1 billion worth of advanced information technology systems. Boehm originated the spiral model, the Constructive Cost Model, and the stakeholder WinWin Spiral Model approach to software management and requirements negotiation.

USC
Center for Systems and Software Engineering
941 W 37th Place
SAL RM 326
Los Angeles, CA 90089-0781
E-mail: boehm@usc.edu

Jo Ann Lane is currently a principal at the USC Center for Systems and Software Engineering conducting research in the area of SoS engineering. In this capacity, she is currently working on a cost model to estimate the effort associated with SoS architecture definition and integration. She is also a part-time instructor, teaching software engineering courses at San Diego State University. Prior to this, Lane was a key technical member of Science Applications International Corporation's Software and Systems Integration Group responsible for the development and integration of software-intensive systems and SoS.

USC
Center for Systems and Software Engineering
941 W 37th Place
SAL RM 328
Los Angeles, CA 90089-0781
E-mail: jolane@usc.edu

Systems Engineering for the Global Information Grid: An Approach at the Enterprise Level

Patrick M. Kern
Deputy to the Assistant Secretary of Defense, Networks and Information Integration/ Department of Defense Chief Information Officer

Because the numerous United States Department of Defense (DoD) and Intelligence Community (IC) networks were originally built to serve many different constituencies, making the Global Information Grid (GIG) a reality requires solving interoperability and performance issues at the enterprise level. This will be accomplished through the use of systems engineering – a discipline whose techniques manage the complexity of systems from abstraction to decomposition. The GIG Technical Foundation (GTF) addresses a number of systems engineering challenges involving focus, evolution, coverage, and applicability.

As you are probably aware, the GIG is a complex, ongoing effort intended to integrate all information systems, services, and applications within the DoD and the IC into a seamless, reliable, and secure network that will support horizontal information flow and net-centric warfare.

The GIG represents a different way of thinking about delivering capabilities, a way of thinking that can cope with the uncertainties we face in the world today. In the past, missions focused on narrow objectives against known adversaries and were organized with tightly managed organizational responsibilities across the DoD and IC constituencies. Today, adversaries are shadowy and shifting, objectives are far-reaching, and new responsibilities link our organizations at all levels. The DoD and IC networks built in the past evolved into stovepipes, tied to missions and organizations that now are forced to adapt to a more fluid world. The GIG confronts uncertainty, inherent in today's world, with the agility that comes from interconnected, interoperable solutions that can be tailored to today's missions and objectives. Making the GIG a reality requires breaking out of stovepipes and solving interoperability and performance issues at the enterprise level. We have approached the problem of building, populating, operating, and protecting the GIG by applying systems engineering discipline to the complex set of communications systems, information systems, services, and applications that make up the GIG. Systems engineering as a discipline provides us with the techniques to manage the complexity of systems.

Enterprise-Wide Systems Engineering (EWSE), as applying systems engineering to the GIG at this level is known, can only succeed by properly focusing the effort. EWSE utilizes interoperability and end-to-end performance as the criteria to determine what is within scope. Enterprise decisions for these requirements are then documented and enforced in the design of GIG component systems, laying the groundwork for the GTF. The GTF is the configuration-managed, synchronized set of all authoritative technical guidance required for planning, developing, acquiring, and implementing an interoperable and secure GIG.

Background

With origins in a wide range of component systems procured to support autonomous agencies and services, the GIG is more accurately an organizing construct than an actual system. Its legacy components vary in terms of performance, storage, and process, and they must continue to support their existing user communities even as they become part of the GIG. While many individual component systems are unknown at the enterprise level, the GIG's component set – as well as the components themselves – will evolve to reflect participant groups' capabilities and financial priorities. The challenge is to establish a process that brings these disparate components together into a single entity that meets the needs of all users.

As GIG component systems are designed, built, and funded by member organizations, it is necessary to deductively establish the functions, protocols, and data models required for their interoperability and performance. Such an investment will benefit all GIG users.

Scope of the Effort

The Assistant Secretary of Defense, Networks and Information Integration (OASD[NII]/DoD Chief Information Officer) tasked the Defense Information Systems Agency to lead an Enterprise Documentation Framework Working Group that would apply systems engineering practices to create the GTF. The GTF provides structure and traceability for all GIG documentation in a manner similar to that of a document tree. The GTF is based on and traceable to operational needs derived from national and DoD strategic guidance and direction. It includes enterprise-level GIG documentation (GIG capabilities; activities; technical requirements, including standards and specifications; and the GIG Architecture) and other GIG-related technical documentation[1].

Applying systems engineering at the enterprise level to support development of the GTF must start with the GIG's vision as outlined in the Net-Centric Operations Environment Joint Capability Document[2]. Once top-level requirements are defined to identify the necessary functionality, this functionality can be decomposed into system segments and sub-segments.

Top-level requirements have been decomposed into three areas – general, enterprise management, and Information Assurance – and flow down to requirements at the segment level. These segments include transport, services, applications, computing infrastructure, and enterprise operations.

Segment and sub-segment requirements are specified as needed for interoperability and performance according to the top-level requirements, which can be traced from GIG capabilities and requirements to segment and sub-segment requirements. Sub-segment requirements, needed to achieve interoperability and end-to-end performance, are often the specification of protocols or mechanisms.

Figure 1 illustrates the relationship between top-level requirements, segment-level requirements, and sub-segment requirements for a transport segment example.

Systems Engineering Challenges

In addition to scope, the GTF addresses a number of systems engineering challenges involving the following:

- **Focus.** All requirements for achieving the GIG capabilities – including what is currently feasible and what requires further development – must be specified by the GTF to ensure that programs understand the needed transitions. Programs, services, and agencies responsible for existing GIG component systems are then responsible for developing transition plans that reflect the requirements of the GTF.

- **Evolution.** Many aspects of the GIG's long-term vision, including pervasive mobility, ad-hoc network connection, efficient resource use, and dynamic resource allocation/management are not achievable through use of current technologies. Long-term GIG design must not be limited to requirements dependent on current technology, but include provisions for emerging and future technologies as well.
- **Coverage.** As mentioned earlier, the GIG is made up of a wide variety of components, many of which are unknown at the enterprise level. Components will be added and removed as organizational needs evolve, and the components themselves will also evolve. As a result, requirements for the GIG must be specified in terms of component type rather than for specific components. Requirements must also be defined for the set of systems needed to meet GIG capabilities rather than for those appropriate only for existing/planned systems.
- **Applicability.** Since GIG users will operate in a variety of environments, requirements need not apply to all environments or modes. Specific domains of applicability must be defined which work in concert to provide overall enterprise capabilities. For example, fixed users are well connected and can reliably reach centralized data centers. The fixed users are not severely constrained in power, memory, storage, and processing. Examples of fixed user modes are camps, posts, stations, and bases served by the Defense Information System Network. Advantaged tactical users operate in a slowly changing environment subject to high latency and limitations on bandwidth that may constrain reach-back to centralized data centers. The advantaged tactical users are not severely constrained in power, memory, storage, and processing. Examples of advantaged tactical user modes are tactical operations centers and Navy ships. Disadvantaged tactical users operate in a highly dynamic topology, with limited and sometimes no fixed infrastructure, subject to disruption in communications and with severe constraints on one or more of power, memory, storage, and processing. An example of disadvantaged tactical user mode is a Mobile Ad-Hoc Network formed by vehicles and dismounted soldiers.

Assembling the GTF

The GTF is intended to address all requirements relating to the GIG's long-term vision, even those not achievable through the use of currently available technologies, protocols, and mechanisms. Sub-segment requirements are divided into two categories – current requirements, which are achievable using current technology, and maturing requirements, which rely on emerging and future technologies.

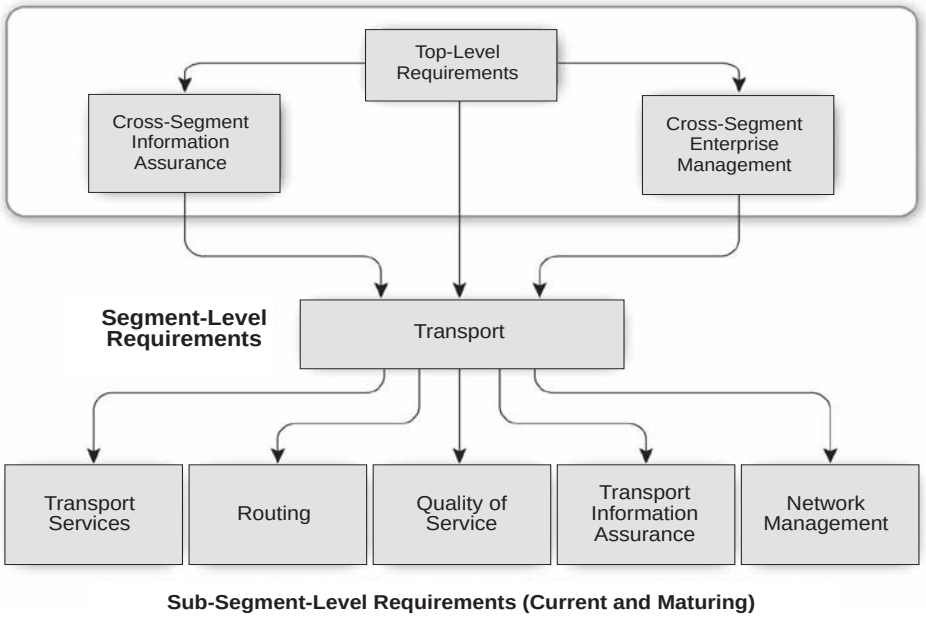
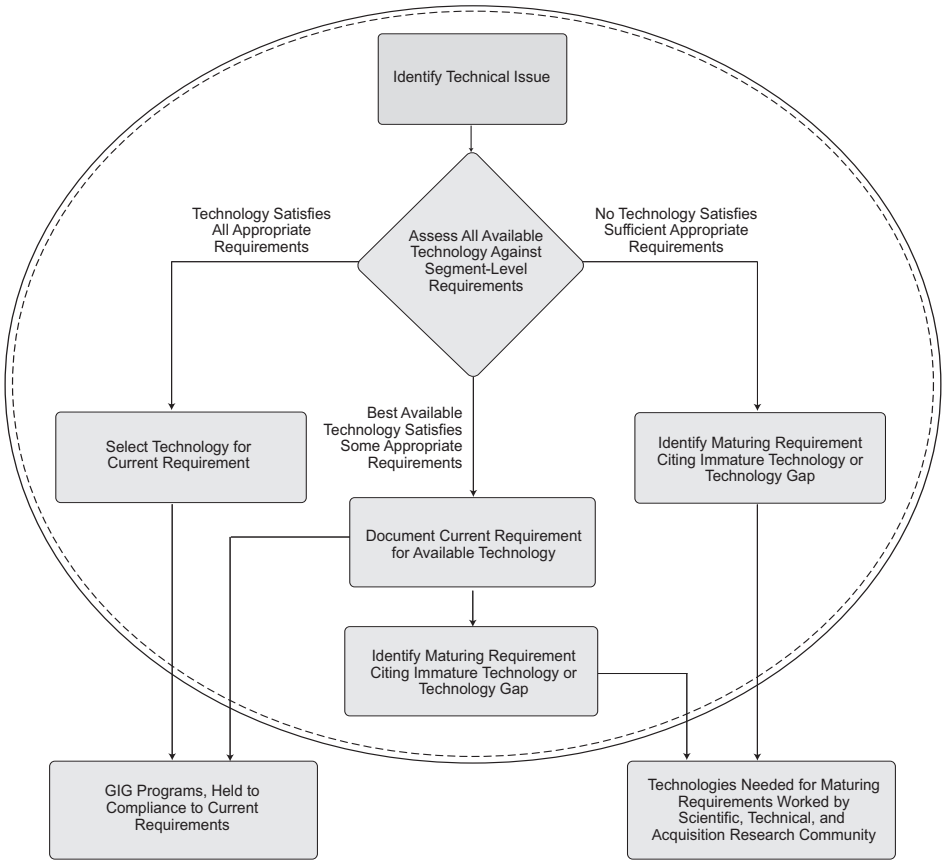


Figure 1: *Transport Segment Example Illustrating the Relationship Between Requirements*

Current requirements are testable and will be enforced in the design of GIG component systems. By contrast, maturing requirements are used to document technologies needed to achieve GIG capabilities, verify the feasibility of achieving GIG capabilities and provide insight on research needed to meet the GIG vision.

Occasionally, use of a technology, mechanism, or protocol that does not satisfy GIG requirements is sanctioned if no other resource is available. In these instances, a current requirement is defined for the existing technology, and a maturing requirement is

Figure 2: *Process for Assessing Technologies for Inclusion in the GTF*



defined for the needed technology. For example, inter-domain routing today would use border gateway protocol version 4 as a current requirement. A new protocol to support pervasive mobility is defined as a maturing requirement.

At all phases of the process of assembling the GTF, stakeholders and subject matter experts participate in working groups to assess technologies and determine the appropriate match for current and maturing requirements. Figure 2 (see page 11) illustrates the process used to assess technologies for inclusion in the GTF.

Community Role in GTF Development

Before the establishment of the GTF, different organizations attempted to define the GIG in separately developed technical, policy, and guidance documents. This resulted in more than 7,000 pages of documentation, which, although well written, contained gaps, overlaps, and inconsistencies that reflected the GIG's fragmented origins in component systems originally intended to function independently. Once it was realized that the emerging GIG documentation did not have the technical maturity to meet end-to-end interoperability and performance compliance standards, members of the GIG user community began to develop baselines against which its individual components could be measured.

Today's GTF is a set of source documents drawn across the GIG community, along with governing statements for GIG development, providing portfolio and program managers with clear guidance on how to implement net-centricity and end-to-end interoperability throughout the acquisition life cycle. It includes authoritative source documents that define the strategic guidance, operational context, operational capabilities, GIG capabilities, GIG activities, and technical direction needed to take the GIG through the following timeframes: near (0-2 years), mid (3-7 years), and far (8+ years).

The GTF also contains governing statements extracted from these source documents that more concisely describe the GIG and are traceable throughout the GIG's Enterprise Document Framework. All content is stored and managed in a DOORS[3] requirements database to facilitate requirements management and configuration control.

Compliance

By developing this integrated approach to compliance assessment that aligns current processes and provides an entry point to the Net-Ready Key Performance Parameter evolution, the GTF does the following:

- Allows program managers to self-assess individual programs.
- Applies consistently to all programs at all levels of oversight.
- Ensures high confidence in end-to-end interoperability and performance compliance at the enterprise level.

Policy has also been revised to direct all compliance to the GTF.

Conclusion

The GIG is an ambitious undertaking that is fundamental to net-centric warfare. We have established an enterprise process to apply systems engineering discipline to the decisions that need to be made to make the GIG a reality. The product of the enterprise approach is a GTF, a new approach to GIG policies and a set of processes for compliance to the GTF.

While the GTF is still an evolving effort, the requirements in the GTF have been flowed into program requirements documents, ensuring more robust interoperability and performance as those programs come online as part of the GIG. The approach we are putting in place will allow us to build, populate, operate and protect the GIG to meet the challenges of today's world.◆

Acknowledgements

The author would like to recognize the contributions and significant input to this article by Ms. Julie Tarr, Senior Systems Engineer, and Mr. Tony Dessimone, Senior Scientist.

Notes

1. Some portions of the GTF are not publicly released.
2. <www.jcs.mil/j6/netcentric.html>.
3. DOORS is an acronym for Dynamic Object-Oriented Requirements System (a Quality Systems and Software, Inc., Quality Systems and Software database management system).

About the Author

Patrick M. Kern is the NII Net-Centric Systems Engineer leading the integration of transformational programs for OASD/NII. He is responsible for end-to-end system engineering for the GIG. Kern has a bachelor's degree in aerospace engineering from the University of Michigan and a master of business administration in engineering management from the University of Colorado.

OASD/NII
3D174 Pentagon
Phone: (703) 697-4704
E-mail: patrick.kern@osd.mil

WEB SITES

Air Force Center for Systems Engineering Reference Library

www.afit.edu/cse/

The Reference Library provides links to key systems engineering policies, guides, industry standards, important historical systems engineering documents, and other related documents.

The United States Army Information Systems Engineering Command

www.hqisec.army.mil/

The United States Army Information Systems Engineering Command has the primary mission of system engineering and integration of information systems for the U.S. Army. Their mission includes the design, engineering, integration, development, sustainment, installation, testing, and acceptance of information systems. It provides matrix support to the program executive officer and program manager structure for systems engineering and integration of assigned information systems.

Systems and Software Engineering Organization Systems Engineering Plan

www.acq.osd.mil/se/as/sep.htm

The Software Engineering Plan is a *living* document that captures a program's current and evolving systems engineering strategy and its relationship with the overall program management effort. Its purpose is to guide all technical aspects of a program, and provides a comprehensive, integrated technical plan to achieve its objectives.

ConOps: The Cryptex to Operational System Mission Success

Alan C. Jost
Raytheon

As engineering firms start to design any number of systems for a variety of customers and end users, the number and variety of system documentation can be overwhelming. Among this pile of documentation, the Concept of Operations (ConOps) stands out as a critically important engineering document that should be created at the beginning of the system development and maintained throughout the engineering life cycle. This article discusses the ConOps and if it truly is necessary in addition to all of the other documentation available.

Of course we need a ConOps! Admittedly, system development programs can be overwhelmed by the number and variety of required documentation. There are system specifications, subsystem specifications, discipline requirements specifications (hardware, software), discipline design documents (hardware, software), interface requirement specifications (IRS), interface control documents (ICD), test plans, procedures, reports, and a multitude other documents that capture what system is to be built, how the system is built, and how the system is tested. There is, however, a critically important engineering document that should be the key document developed at the beginning of the system development and maintained throughout the engineering life cycle. What is this critically important engineering document, the ConOps? It is key to successfully developing an operational system. This article covers ConOps – what it is, what it is not, and its contents. I will also show the importance of the ConOps in three situations dealing with four Air Force programs: the Over-the-Horizon Backscatter (OTH-B) Radar, the Seek Score Radar Bomb Scoring System, and the PAVE[1] Phased Array Warning System (PAWS) Ballistic Missile Early Warning System (BMEWS).

The ConOps is a descriptive document usually created by the future operational users of the system. It details what the system is going to be used for, what other systems it will be used with and communicate with, what kind of data and information it requires and supplies, how it is going to be used by the operational user, who is going to be the operational user, how it is going to get to where it is going to be used, and how it is going to be maintained. In the past, the document has been called a variety of titles (e.g., ConOps, CONOPS, and Mission Needs Statement); regardless, in this article it will be called the ConOps. In my many years in the Air Force and in industry, I have found the ConOps to be one of the most difficult engineering documents to write. Why? The ConOps is a description on how the system is going to be used. It is not an engineering document that details system requirements or describes the desired design of the system. The ConOps should be written devoid of system requirement statements and engineering design. This is the cornerstone document that drives the follow-on engineering documents, where requirements flow out of the ConOps into the system-level requirements specifications during the system requirements analysis life-cycle phase. The difficulty is keeping the requirements and design from creeping into the ConOps. The ConOps should not constrain the engineering process and its creativity in solving the operational needs of the operational users. At times, however, writing the ConOps creates a dilemma for the operational command who wish to dictate not only how the system is to be used, but how the system should be built. Who should write this critically important document, the ConOps?

Who Should Author the ConOps?

Naturally, the optimum authors of the system should be the operational users of the system. This poses a problem for a number of reasons. In some instances with the extensively long procurement and system development life-cycle time frames, the operational users of the system may not even be old enough to be in the military or in the employment pool when the ConOps needs to be initially written. Then you have the current operational users who are extremely busy performing their operational duties and have very little time to devote to writing a detailed ConOps. Some operational commands have organizations within the command to generate ConOps and future systems requirements to meet their command's operational missions. These organizations usually have a staff mix of recent operational users and engineers. In my experience, these folks are usually very passionate over improving their organization's capability in the field. They want to make it much, much easier for their future operational users – their comrades in arms. It is, however, difficult to write the ConOps without trying to drive the requirements or design of the system. Operational commands and product divisions that procure the systems usually hire engineering organizations like MITRE or system engineering firms to assist in developing the required engineering documents, and yes, even a ConOps. Again in my experience the ConOps generated by these organizations include system requirements and engineering design influences. But the operational users will contend that if they don't have the system design, it is difficult for them to write the ConOps. Likewise, not having the ConOps is a constraining factor in coming up with the engineering requirements and design – the classic chicken or the egg dilemma.

What Is a ConOps?

The Institute of Electrical and Electronics Engineers (IEEE) Std. 1362-1998 Guide for Information Technology – System Definition – ConOps Document Description provides user organizations a way to describe their missions and organizational objectives to contractors from an integrated systems point of view. The document abstract reads as follows:

> The format and contents of a concept of operations (ConOps) document are described. A ConOps is a user-oriented document that describes system characteristics for a proposed system from the users' viewpoint. The ConOps document is used to communicate overall quantitative and qualitative system characteristics to the user, buyer, developer, and other organizational elements (for example, training, facilities, staffing, and maintenance). It is used to describe the user organization(s), mission(s),

and organizational objectives from an integrated systems point of view. [1]

The purpose of the ConOps is to provide the user community a vehicle for describing their operational needs that must be satisfied by the system under development.

The ConOps approach provides an analysis activity and a document that bridges the gap between the user's needs and visions and the developer's technical specifications. In addition, the ConOps document provides the following:

- A means of describing a user's operational needs without becoming bogged down in detailed technical issues that shall be addressed during the systems analysis activity.
- A mechanism for documenting a system's characteristics and the user's operational needs in a manner that can be verified by the user without requiring any technical knowledge beyond that required to perform normal job functions.
- A place for users to state their desires, visions, and expectations without requiring the provision of quantified, testable specifications. For example, the users could express their need for a *highly reliable* system and their reasons for that need without having to produce a testable reliability requirement. (In this case, the user's need for *high reliability* might be stated in quantitative terms by the buyer prior to issuing a request for proposal [RFP], or it might be quantified by the developer during requirements analysis. In any case, it is the job of the buyer and/or the developer to quantify users' needs [and not the responsibility of the user even though they are usually very anxious to provide the ole 0.99999 reliability number instead of *highly reliable*.])
- A mechanism for users and buyer(s) to express thoughts and concerns on possible solution strategies. In some cases, design constraints dictate particular approaches. In other cases, there may be a variety of acceptable solution strategies. The ConOps document allows users and buyer(s) to record design constraints and the rationale for those constraints as well as indicate the range of acceptable solution strategies.[1]

Structure of the ConOps

By examining the IEEE's suggested ConOps structure, you can see how it is oriented around the operational user's needs. It is not a simple document to write and complete without having system requirements and design creep into the document; try to describe something as common as your next dream car without including system requirements or design in your dream car ConOps [1]. As you can see, the contents of an IEEE compliant ConOps document is defined in Section 4 of the IEEE Standard 1362-1998 – go ahead try to write one for your new dream car.

- Section 1: Scope.
- Section 2: References.
- Section 3: Definitions.
- Section 4: Elements of a ConOps document.
  - 4.1 Scope (Clause 1 of the ConOps document).
  - 4.2 Referenced documents (Clause 2 of the ConOps document).
  - 4.3 Current system or situation (Clause 3 of the ConOps document) .
  - 4.4 Justification for and nature of changes (Clause 4 of the ConOps document).
  - 4.5 Concepts for the proposed system (Clause 5 of the ConOps document).
  - 4.6 Operational scenarios (Clause 6 of the ConOps document).
  - 4.7 Summary of impacts (Clause 7 of the ConOps document).
  - 4.8 Analysis of the proposed system (Clause 8 of the ConOps document).
  - 4.9 Notes (Clause 9 on the ConOps document).
  - 4.10 Appendices (Appendices of the ConOps document).
  - 4.11 Glossary (Glossary of the ConOps document) [1].

Joint Authorship of the ConOps

While potentially creating blasphemy, I suggest the critically important ConOps be drafted by an operational command with as much operational detail as possible and included in the RFP during the initial program phase. The initial ConOps may have sections 4.1-4.4, 4.6-4.7, and 4.9-4.11. The contractor can input section 4.5 and 4.8 in their proposal. But once the contract is awarded, all parties should finalize the initial ConOps for the system under development. If it is impossible to eliminate any requirements or design content from the ConOps – at least put them in the context of suggestions. The initial effort in the engineering life cycle should be for the contractor, product division, support contractors, and operational command to jointly update and finalize the draft ConOps found in the RFP. The result is to move any system requirements from the ConOps into the associated system specification and move further detailed requirements in lower-level specifications and design documents. Also, while you want the ConOps to remain relatively stable and unchanging, the reality of the engineering life cycle is that it does take a long time to engineer and develop these systems. In the meantime, the operational mission may change and, therefore, the ConOps should be updated to reflect the current operational mission. This should involve all participants in the engineering and development of the system so that at the end of the day, not only did the contractor build the system right (i.e. met all the system requirements), but also built the right system (i.e. met all the operational user's needs). As a matter of fact, these are the three process areas in the Capability Maturity Model® Integration (CMMI®) Maturity Level 3 process areas of Verification (build the system right) and Validation (build the right system). More to the point, the Technical Solution process area has a specific practice for *evolving operational concepts and scenarios* [2].

System Requirements Continue to Reflect the ConOps

So, one of the important aspects of the engineering process is to make sure that not only are you meeting the system requirements, but that those system requirements actually reflect the operational mission of the end user. This is reflected in the CMMI model which was collaboratively written by some very smart folks in industry and government [2]. Taken as industry best practices and extensive lessons learned, one can conclude it is extremely important to keep the ConOps and system requirements relatively in sync throughout the engineering life cycle. Naturally, one would expect that the mission operational needs would match the system requirements maintained in the system specification. However, with extended procurement schedules and restricted budgets, the need to field a system that meets some or most of the requirements sometimes takes over the procurement process. If the operational mission changes, this usually results in changes to the requirements. Of course, depending on when in the development life cycle these changes occur, the cost of the resulting Engineering Change Proposal (ECP) can be very expensive

® Capability Maturity Model and CMMI are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

and, therefore, the decision is made that the contract specification is *not* updated and the ECPs are not made to the contract to match the operational requirements. If many of these mission changes are not incorporated into the contract via ECP, you will have the situation where the system being built is starting to drift away from being the system needed by the operational user. Once the ConOps is developed, it is critical to make sure that any changes made to the operational mission or needs are translated into updated system requirements. These changes need to be incorporated into the systems contract. Let us examine an example of where the system's operational mission changed rather dramatically from the original ConOps, resulting in an operational test that was less than satisfactory. In addition, let us examine how creative use of new ConOps can lead to renewed use of systems whose original ConOps were no longer valid but were successfully reused after readjusting the systems' original ConOps.

OTH-B Radar

The Army-Navy Fixed Radar Search 118 (AN/FPS-118) OTH-B was produced for the Electronic Systems Division of Air Force Systems Command to fill a vital need for long-range air surveillance for North America. Designed by General Electric in the 1980s, the AN/FPS-118 would provide detection and tracking of airborne threats at ranges up to 1,800 nautical miles regardless of altitude. The Air Force's OTH-B air defense radar system is, by several criteria, the largest radar system in the world. Six OTH-B radars see far beyond the range of conventional microwave radars by bouncing their radar waves off the ionosphere, an ionized layer about 200 km above the earth. It was developed over 25 years at a cost of $1.5 billion to warn against Soviet bomber attacks when the planes were still hundreds of miles from U.S. airspace [3].

With the end of the Cold War, just months after their deployment, the three OTH radars on the West Coast were mothballed, the Central and the incomplete Alaskan Systems were cancelled, but the three radars in Maine were redirected to counter-narcotics surveillance. In 1994, Congress directed the Air Force to continue operating the East Coast OTH-B radar. The East Coast formally ceased OTH-B operations in October 1997 [3].

Here was this new radar system that used the novel idea of using the ionosphere as part of a radar system that could literally see over the horizon to get an advanced warning of incoming Soviet Union aircraft that carried cruise missiles. Because of the novel use of the ionosphere as a component of the radar, it took a long time (25 years) to develop the radar system and its software. So long in fact, that in the meantime the Soviet Union collapsed, and the threat of the Soviet aircraft with cruise missile disappeared; this occurred literally months before the OTH-B radar system was to undergo its operational testing for its operational mission. The operational command that originally contracted for the OTH-B Radar system no longer had an operational need for the system since the original threat had disappeared.

The ConOps for the OTH-B Radar changed dramatically as the operational usage of the radar system shifted to a Drug Enforcement Agency (DEA) use of the radar to monitor potential drug trafficking off the East Coast of the United States. The Air Force Operational Test and Evaluation Center (AFOTEC) were responsible for conducting the operational tests of the radar, and were using the current DEA operational mission ConOps as the guidance for developing the operational test procedures. The dramatic shift of the operational mission from the Air Force to the DEA occurred almost at the end of the development life cycle with minimal chance to change the ConOps and then go through the ECP process to change the system specification along with all the associated changes that would be needed in the system software. It is important to note, the contractor with the product division conducts a series of tests to prove that the system was built right, i.e. it passes all the system requirements or the *shalls* in the system specification, which it did with flying colors. However, the mission of the operational testing organization is to test the system against the current ConOps to make sure the system that was built was the right system to support its current operational mission. Naturally, the operational system test did not go so well since the ConOps changes could not generate adjustments to requirements at the very end of the program. This was not the contractor's fault whatsoever, but it shows an example in the extreme of what could happen when the changes in the ConOps are not reflected in the system requirements on contract. This is a dramatic example that is atypical in a discussion on the importance of a current ConOps in sync with the contract documents and system requirements. Let us examine two systems where their original ConOps were adjusted to allow the systems renewed and different missions for the operational command.

Radar Bomb Scoring System

Before sophisticated laser guided bombs, we used ground directed radar guided bombing, where the radar system would direct the flight path of the bomber aircraft to a drop point in the sky. With computer algorithms using aircraft location, meteorological inputs, and flight characteristics of the weapon, the operators of the radar system could indicate to the pilots and weapons controllers when and where to release the weapon to hit the selected targets. The operational command generated a ConOps that was flexible enough to allow the reuse of the Ground Directed Bombing System (GDBS) to be converted into a Radar Bomb Scoring System, known as SEEK SCORE. The AN/TPQ-43 SEEK SCORE is an automatic tracking radar system. This system replaced the antiquated AN/MSQ-46 and AN/MSQ-77 Bomb Directing Central systems used during the Vietnam conflict to guide bombers to their target. The SEEK SCORE AN/TPQ-43 can automatically *score* accuracy of simulated bomb releases electronically. Using computer targeting coordinates the SEEK SCORE computer performs a complete ballistics computation on any type of simulated weapon release from where the tracked aircraft is at the release point to where the *target* is. This computation provides an accurate miss distance score. The radar system can also perform a comparison of aircraft position in relation to a target to *score* the navigation and timing accuracy of an aircrew. The computerized scoring capabilities of the SEEK SCORE enhances USAF training because an aircrew can practice flying over any type of terrain at any altitude and practice bomb drops or navigation without ever dropping bombs. This clever reuse of the existing radar system capability was directly related to a ConOps that was void of design and requirements, allowing for the reuse of the GDBS system to support the new operational use of the system as a Radar Bomb Scoring System [4].

PAVE PAWS to BMEWS Site II

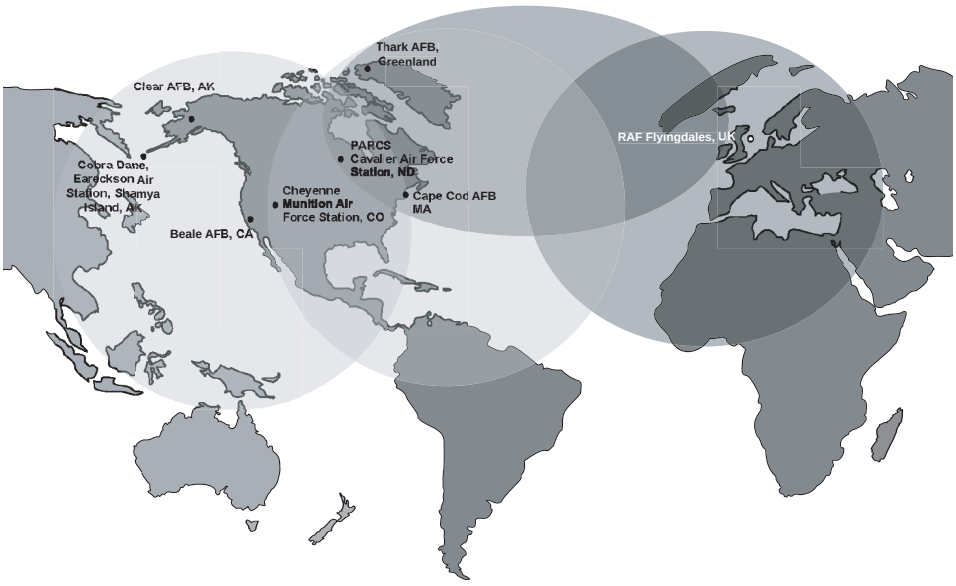
At height of the cold war, the Air Force built six extremely large phased array radar systems (see Figure 1, page 16) [5]. One of the systems known as PAVE (PAVE PAWS) initially had large phased array radars located in four locations in the con-

tinental United States, the first of which was located at Otis Air Force Base on Cape Cod Massachusetts. The design of the Cape Cod system had a dual phased array radar face providing radar coverage eastward over the Atlantic Ocean. In parallel, the same operational command also needed to upgrade the existing BMEWS that was located in three places: Thule Greenland; Fylingdales, United Kingdom; and Clear Air Force Base, Alaska. The PAVE PAWS mission was to provide warning of a Sea-Launched Ballistic Missile (SLBM) attack against the United States, while the BMEWS system was focused on Inter-Continental Ballistic Missile attack against North America and United Kingdom. The original BMEWS system built in the early 1960s really needed to be refurbished and upgraded in the 1980s and 1990s. However, since the operational command stated its mission needs in an operational context rather than specific design and requirements, the contractor building PAVE PAWS phased array radars offered to upgrade the current BMEWS by replacing them with a PAVE PAWS-like, two-faced, phased array radar built on the existing BMEWS buildings at the Thule Site with adjusted tracking and reporting software. Known as the BMEWS Radar Upgrade Site I (BMEWS I), the new radar system provided a dramatically enhanced capability. The next BMEWS site upgraded was the Fylingdales, United Kingdom site, but with pyramid-shaped, three-faced phased array radar system. Again with the ConOps based on mission needs, the Fylingdales system could extend the two-faced, phased array system installed at Thule to handle the mission at Fylingdales. To take this concept to the extreme, the Clear AFB, Alaska BMEWS Site II (BMEWS II) was recently upgraded by actually dismantling the PAVE PAWS site located at Warner-Robbins Air Force Base and reinstalling the radar system at the Clear location. Naturally, the processing equipment and software also needed upgrading since the system was going from the SLBM mission to its current mission, but the same level of flexibility in the ConOps allowed for extensive reuse of existing systems for new operational missions.

Summary

The purpose of this article is to give a viewpoint on how extremely important the ConOps document is in the system engineering and development life cycle of the system. Not only is the document important in the beginning of the life cycle, but it needs to be revisited during the major lifecycle phase points of the program to ensure the program remains on track to build the right system. It is also important to make sure it is updated to make sure the ever explosive growth of technology is continually examined to see how technology insertion can be accomplished at the most economical point of the engineering life cycle. While the OTH-B situation is in the extreme, it goes to the point that there are two aspects of the system life cycle: Did the contractor build the system right (all *shalls* passed) and did the contractor build the right system (meets the operational missions)? We saw two significant systems that, by having a mission-oriented ConOps, allowed the contractors to bring creativity, flexibility, and cost-savings reuse of existing systems for new missions, such as PAVE PAWS for BMEWS and the new SEEK SCORE radar bomb scoring system from the old, existing GDBS.◆

Figure 1: *PAVE PAWS and BMEWS Coverage Map*



References

1. IEEE. IEEE Guide for Information Technology – System Definition-Concept of Operations Document. Std. 1362-1998. IEEE Electronic Library.
2. Capability Maturity Model Integrated (CMMI), Systems and Software: V1.1, Software Engineering Institute, 2001.
3. Air Combat Command. "Fact Sheet: Over-the-Horizon Backscatter Radar: East and West." U.S. Air Force Fact Sheet <www.acc.af.mil/factsheets/factsheet _print.asp?fsID3863&page=1>.
4. FAS Military Analysis Network <www.fas.org/man/dod-101/sys/ac/equip/an-tpq-43.htm>.
5. PAVE PAWS. Wikipedia <http://en.wikipedia.org/wiki/PAVE_PAWS>.

Note

1. While many people tried to create an acronym meaning for PAVE, it was never an acronym for anything, it simply meant an Air Force Program.

About the Author

Alan C. Jost (Lt. Col., U.S. Air Force, retired) is a senior software program manager in the Raytheon Northeast Software Engineering Center (SWEC), where he works multiple tasks including the Software Engineering Program Group Executive Committee for SWEC's CMM/CMM Integration Level 5 sustainment and as a process engineer on the AutoTrac III Air Traffic Control Product Line and DD(X) ExComms Software Cross Product Team. Jost joined Raytheon after 20 years in the Air Force. He has served in a variety of line management, process engineering, and software task management roles in SWEC.

Raytheon
MS 3-1-3914
1001 Boston Post RD
Marlborough, MA 01752
Phone: (508) 490-4282
Fax: (508) 490-1366
E-mail: alan_c_jost @raytheon.com

Software System Engineering: A Tutorial

Dr. Richard Hall Thayer
Software Management Training LLC

Applying system-engineering principles specifically to the development of large, complex software systems provides a powerful tool for process and product management. Software engineering has its early roots in system engineering which is reflected in their many common terms. This article discusses a merger between system engineering and software engineering called software system engineering. System engineering looks at controlling the total system development including software. Software engineering looks at controlling just software development. (System engineers would call software engineering component engineering.) The application of system engineering to the development of software gives a large measure of control software development.

Applying system engineering principles specifically to the development of large, complex software systems provides a powerful tool for process and product management. This process is called software engineering. Dr. Winston Royce, father of the Waterfall chart, points out that software engineering was developed from system engineering, and he argued for calling the union software system engineering. Unfortunately, this did not stick, and software engineering and software system engineering can be viewed as separate processes. Software systems have become larger and more complex than ever. We can attribute some of this growth to advances in hardware performance – advances that have reduced the need to limit a software system's size and complexity as a primary design goal. Microsoft Word is a classic example: A product that would fit on a 360-kilobyte diskette 20 years ago now requires a 600-megabyte compact disc.

But there are other reasons for increased size and complexity. Specifically, software has become the dominant technology in many if not most technical systems. It often provides the cohesiveness and data control that enable a complex system to solve problems.

Figure 1 is a prime example of this concept. In an air traffic control system, software connects the airplanes, people, radar, communications, and other equipment that successfully guide an aircraft to its destination. When the Federal Aviation Administration systems were upgraded to automation back in the 1960s, the much larger systems could handle many more aircraft over a larger terrain. However, these larger systems continued to use much of the earlier 1950s hardware; it was the software that enabled larger groups of hardware to work together towards the common goal of safely delivering an aircraft from takeoff to landing. Software provides the system's major technical complexity.

Because of this increase in size and complexity, the vast majority of large software systems do not meet their projected schedule or estimated cost, nor do they completely fulfill the system acquirer's expectations[1]. This phenomenon has long been known as the software crisis [1]. In response to this crisis, software developers have introduced different engineering practices into product development.

As large system solutions become increasingly dependent on software, a system engineering approach to software development can help avoid the problems associated with the software crisis.

Simply tracking a development project's managerial and technical status – resources used, milestones accomplished, requirements met, and tests completed – does not provide sufficient feedback about the project's health. Instead, we must manage the *technical processes* as well as its products. System engineering provides the tools the technical management task requires.

The application of system engineering principles to the development of a computer software system produces activities, tasks, and procedures called software system engineering (SwSE). Many practitioners consider SwSE to be a special case of system engineering and others consider it to be part of software engineering. However, it can be argued that SwSE is a distinct and powerful tool for managing the technical development of large software projects.

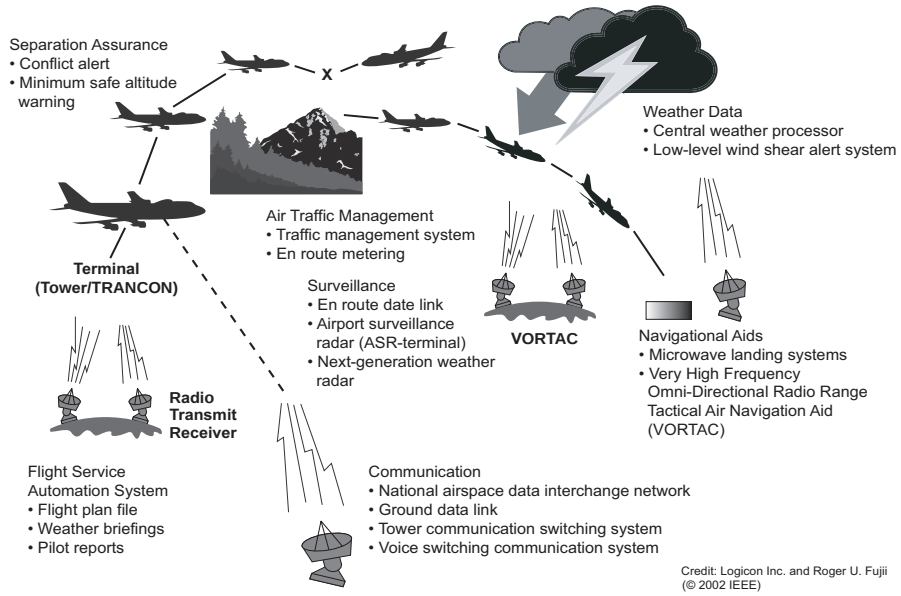
This tutorial integrates the definitions and processes from the Institute of Electrical and Electronics Engineers (IEEE) software engineering standards [2] into the SwSE process.

Systems and System Engineering

A *system* is a collection of elements related in a way that allows a common objective to be accomplished. In computer systems, these elements include hardware, software, people, facilities, and processes.

System engineering is the practical appli-

Figure 1: *Software Ties the System Together*



Credit: Logicon Inc. and Roger U. Fujii
(© 2002 IEEE)

Portions of this article appeared in Computer, Apr. 2002. © 2003 IEEE. Reprinted with permission from *Computer*, Vol.35, Issue 4, pp. 68-73, Apr. 2002.

cation of scientific, engineering, and management skills necessary to transform an operational need into a description of a system configuration that best satisfies that need. It is a generic problem-solving process that applies to the overall *technical* management of a system development project. This process provides the mechanism for identifying and evolving a system's product and process definitions.

IEEE Std. 1220-1998 describes the system engineering process and its application throughout the product life cycle [3]. *System engineering produces documents, not hardware.* These documents are associated with the developmental processes within the project's life-cycle model. They also define the expected process environments, interfaces, products, and risk management tools throughout the project.

System engineering involves five functions:

- *Problem definition* determines the needs and constraints through analyzing the requirements and interfacing with the acquirer.
- *Solution analysis* determines the set of possible ways to satisfy the requirements and constraints, analyzes the possible solutions, and selects the optimum one.
- *Process planning* determines the tasks to be done, the size and effort to develop the product, the precedence between tasks, and the potential risks to the project.
- *Process control* determines the methods for controlling the project and the process, measures progress, reviews intermediate products, and takes corrective action when necessary.
- *Product evaluation* determines the quality and quantity of the delivered product through evaluation planning, testing, demonstration, analysis, examination, and inspection.

System engineering provides the baseline for all project development, as well as a mechanism for defining the *solution space*. The solution space describes the product at the highest level – before the system requirements are partitioned into the hardware and software subsystems.

This approach is similar to the software engineering practice of specifying constraints as late as possible in the development process. The further into the process a project gets before defining a constraint, the more flexible the implemented solution will be.

What Is SwSE?

The term *software system engineering* dates from the early 1980s and is credited to Dr. Winston Royce [4], an early leader in software engineering. SwSE is responsible for the overall technical management of the system and the verification of the final system products. As with system engineering, SwSE produces documents, not components. This differentiates it from software engineering, which produces computer programs and user manuals.

SwSE begins after the system requirements have been partitioned into hardware and software subsystems. SwSE establishes the baseline for all project software development. Like software engineering, it is both a technical and a management process. The SwSE technical process is the analytical effort necessary to transform user operational needs into the following:

- A software system description.
- Software system requirements and design specifications.
- Necessary procedures to verify, test, and accept the finished software product.
- Necessary documentation to use, operate, and maintain it.

SwSE is not a job description. It is a process that many people and organizations perform: system engineers, managers, software engineers, programmers, and – not to be ignored – acquirers and users.

Software developers often overlook system engineering and SwSE in their projects. They consider systems that are all software or that run on commercial off-the-shelf (COTS) computers to be just software projects, not system projects. Ignoring the systems aspects of software development contributes to our long-running software crisis.

SwSE and Software Engineering

Early in my software engineering career, I was informed that software engineering was the engineering of software copied from the hardware engineers (e.g. electrical engineers, mechanical engineers, and so forth). I was well acquainted with the mechanics of software engineering which made it different from computer science. The following are examples of what makes the mechanics of software engineering different than computer science:

- Dividing the project into phases such as life-cycle development methods.
- Managing software as a separate project.
- Using intermediate products (specifications), e.g., requirements specifications, design specifications.
- Reviewing, testing, and auditing.
- Using configuration management and quality (process) assurance.
- Prototyping and the reuse of existing components.

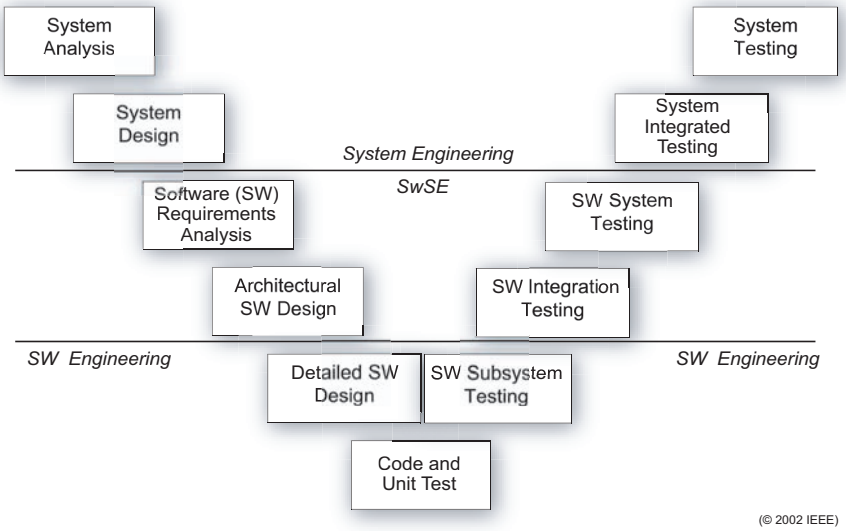
Later, I had an opportunity to mingle with a number of engineers from a conventional engineering discipline, and I asked them about some of our software engineering processes such as those listed above. Most had no idea what I was referring to.

Then by chance, I had an opportunity to work for a company that used system engineering. I then realized that I had found the source of software engineering processes: It was system engineering.

Both SwSE and software engineering are technical and management processes, but software engineering produces software components and their supporting documentation. Specifically, software engineering is the following:

- The practical application of computer science, management, and other sci-

Figure 2: *Engineering Relationships*



ences to the analysis, design, construction, and maintenance of software and its associated documentation.
- An engineering science that applies the concepts of analysis, design, coding, testing, documentation, and management to the successful completion of large, custom-built computer programs under time and budget constraints.
- The systematic application of methods, tools, and techniques that achieve a stated requirement or objective for an effective and efficient software system.

Figure 2 illustrates the engineering relationships between system engineering, SwSE, and software engineering. Traditional system engineering does initial analysis and design as well as final system integration and testing.

During the initial stage of software development, SwSE is responsible for software requirements analysis and architectural design. SwSE also manages the final testing of the software system *component engineering*.

SwSE and Project Management

The project management process involves assessing the software system's risks and costs, establishing a schedule, integrating the various engineering specialties and design groups, maintaining configuration control, and continuously auditing the effort to ensure that the project meets costs and schedules and satisfies technical requirements [5].

Figure 3 illustrates the management relationships between project management, SwSE, and software engineering. Project management has overall management responsibility for the project and the authority to commit resources. SwSE determines the technical approach, makes technical decisions, interfaces with the technical acquirer, and approves the final software product. Software engineering is responsible for developing the software design, coding the design, and developing software components.

The Functions of SwSE

Table 1 lists the five main functions of system engineering correlated to SwSE, along with a brief general description of each SwSE function.

Requirements Analysis

The first step in any software development activity is to determine and document the system-level requirements in either a system requirements specification (SRS) or a software requirements specification or both. Software requirements include capabilities that a user needs to solve a problem or achieve an objective as well as capabilities that a system or component needs to satisfy a contract, standard, or other formally imposed document [6].

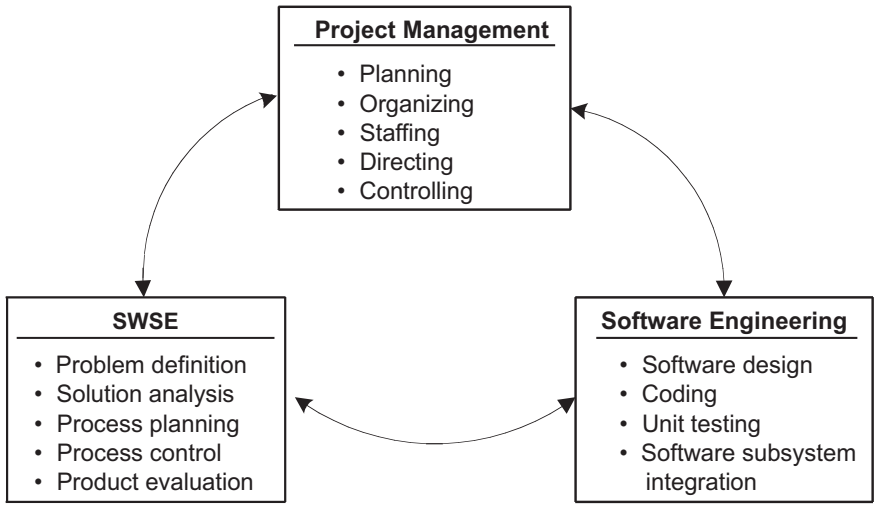


Figure 3: *Management Relationships*

We can categorize software requirements as follows [7]:
- *Functional requirements* specify functions that a system or system component must be capable of performing.
- *Performance requirements* specify performance characteristics that a system or system component must possess such as speed, accuracy, and frequency.
- *External interface requirements* specify hardware, software, or database elements with which a system or component must interface, or set forth constraints on formats, timing, or other factors caused by such an interface.
- *Design constraints* affect or constrain the design of a software system or software system component, for example, language requirements, physical hardware requirements, software development standards, and software quality assurance standards.
- *Quality attributes* specify the degree to which software possesses attributes that affect quality, such as correctness, reliability, maintainability, and portability.

Software requirements analysis begins after system engineering has defined the acquirer and user system requirements. Its functions include identification of all – or as many as possible – software system requirements, and its conclusion marks the established requirements baseline, sometimes called the allocated baseline.

Software Design

Software design is the process of selecting and documenting the most effective and efficient system elements that together will implement the software system requirements [8]. The design represents a specific, logical approach to meet the software requirements.

Software design is traditionally partitioned into two components:

Table 1: *System Engineering Functions Correlated to SwSE*

System Engineering Function	SwSE Function	SwSE Function Description
Problem Definition	Requirements Analysis	Determine needs and constraints by analyzing system requirements allocated to software.
Solution Analysis	Software Design	Determine ways to satisfy requirements and constraints, analyze possible solutions, and select the optimum one.
Process Planning	Process Planning	Determine product development tasks, precedence, and potential risks to the project.
Process Control	Process Control	Determine methods for controlling project and process, measure progress, and take corrective action where necessary.
Product Evaluation	Verification, Validation, and Testing (VV&T)	Evaluate final product and documentation.

(© 2002, IEEE)

SwSE Planning Activities	Project Management Planning Activities
Determines tasks to be done.	Determines skills necessary to do the tasks.
Establishes order of precedence between tasks.	Establishes schedule for completing the project.
Determines size of the effort.	Determines cost of the effort (in staff time).
Determines technical approach to solving the problem.	Determines managerial approach to monitoring the project's status.
Selects analysis and design tools.	Selects planning tools.
Determines technical risks.	Determines management risks.
Defines process model.	Defines process model.
Updates plans when the requirements or development environment change.	Updates plans when the managerial conditions and environment change.

Table 2: *Process Planning Versus Project Planning*

- *Architectural design* is equivalent to system design, during which the developer selects the system-level structure and allocates the software requirements to the structure's components. Architectural design – sometimes called *top-level design* or *preliminary design* – typically defines and structures computer program components and data, defines the interfaces, and prepares timing and sizing estimates. It includes information such as the overall processing architecture, function allocations (but not detailed descriptions), data flows, system utilities, operating system interfaces, and storage throughput.
- *Detailed design* is equivalent to component engineering. The components in this case are independent software modules and artifacts.

The methodology proposed here allocates architectural design to SwSE and detailed design to software engineering.

Process Planning

Planning specifies the project goals and objectives and the strategies, policies, plans, and procedures for achieving them. It defines in advance what to do, how to do it, when to do it, and who will do it.

Planning a software engineering project consists of SwSE management activities that lead to selecting a course of action from alternative possibilities and defining a program for completing those actions.

There is an erroneous assumption that project management performs all project planning. In reality, project planning has two components – one accomplished by project management and the other by SwSE – and the bulk of project planning is an SwSE function. (This is not to say that project managers might not perform both functions.)

Table 2 shows an example partitioning of planning functions for a software system project.

Process Control

Control is the collection of management activities used to ensure that the project goes according to plan. Process control measures performance and results against plans, notes deviations, and takes corrective actions to ensure conformance between plans and actual results.

Process control is a feedback system for how well the project is going. Process control asks questions such as the following: Are there any potential problems that will cause delays in meeting a particular requirement within the budget and schedule? Have any risks turned into problems? Is the design approach still doable?

Control must lead to corrective action – either bringing the status back into conformance with the plan, changing the plan, or terminating the project.

Project control also has two separate components: control that project management accomplishes and control that software systems engineering accomplishes. Table 3 shows an example partitioning of control functions for a software system project.

Table 3: *Process Control Versus Project Control*

SwSE Control Activities	Project Management Control Activites
Determines the requirements to be met.	Determines the project plan to be followed.
Selects technical standards to be followed, for example, IEEE Std. 830 [7].	Selects managerial standards to be followed, for example, IEEE Std. 1058 [5].
Establishes technical metrics to control progress, for example, requirements growth, errors reported, or rework.	Establishes management metrics to control progress, for example, cost growth schedule slippage, or staffing shortages.
Uses peer reviews, in-process reviews, software quality assurance, VV&T, and audits to determine adherence to requirements and design.	Uses joint acquirer-developer (milestone) reviews and software configuration management to determine adherence to cost, schedule, and progress.
Re-engineers the software requirements when necessary.	Replans the project plan when necessary.

VV&T

The *VV&T* effort determines whether the engineering process is correct and the products are in compliance with their requirements [9]. The following critical definitions apply:

- *Verification* determines whether the products of a given phase of the software development cycle fulfill the requirements established during the previous phase. Verification answers the question, *am I building the product right?*
- *Validation* determines the correctness of the final program or software with respect to the user's needs and requirements. Validation answers the question, *am I building the right product?*
- *Testing* is the execution of a program or partial program, with known inputs and outputs that are both predicted and observed for the purpose of finding errors. Testing is frequently considered part of validation.

Verification and Validation (V&V) is a continuous process of monitoring system engineering, SwSE, software engineering, and project management activities to determine that they are following the technical and managerial plans, specifications, standards, and procedures. V&V also evaluates the software engineering project's interim and final products. Interim products include requirements specifications, design descriptions, test plans, and review results. Final products include software, user manuals, training manuals, and so forth.

Any individual or function within a software development project can do V&V. SwSE uses V&V techniques and tools to

evaluate requirements specifications, design descriptions, and other interim products of the SwSE process. It uses testing to determine if the final product meets the project requirements specifications.

The last step in any software development activity is to validate and test the final software product against the software requirements specification and to validate and test the final system product against the SRS. System engineering and SwSE are disciplines used primarily for technical planning in the front end of the system life cycle and for verifying that the plans were met at the project's end. Unfortunately, a project often overlooks these disciplines, especially if it consists entirely of software or runs on COTS computers.

Summary and Conclusions

Ignoring the systems aspects of any software project can result in software that will not run on the hardware selected or will not integrate with other software systems.

Conducting *software engineering* without conducting SwSE puts a project in jeopardy of being incomplete or having components which do not work together, and/or exceeding the project's scheduled budget.

Software engineering and SwSE are primarily disciplines used in the front end of the system life cycle for technical planning and at the very late part of the life cycle to verify if the plans have been met. A review of the emphasis in this article will show that much of the work of planning and SwSE is done during the top-level requirements analysis and top-level design phases. The other major activity of SwSE is the final validation and testing of the completed system.

Software engineering principles, activities, tasks, and procedures can be applied to software development. This article has summarized, in broad steps, what is necessary to implement SwSE on either a hardware-software system (that is primarily software) or on an almost total software system. SwSE is not cheap, but it is cost effective.◆

References

1. Gibbs, W.W. "Software's Chronic Crisis." Scientific American Sept. 1994: 86-95.
2. IEEE. Software Engineering Standards Collection. Vol. 1-4. Piscataway: IEEE Press, 1999.
3. IEEE. Standard for Application and Management of the System Engineering Process. Std. 1220-1998, Piscataway: IEEE Press, 1998.
4. Royce, W.W. "Software Systems Engineering." Management of Software Acquisition. Fort Belvoir, VA: Defense Systems Management College, 1981-1988.
5. IEEE. Standard for Software Project Management Plans. Std. 1058-1998. Piscataway: IEEE Press, 1998.
6. IEEE. Standard Glossary of Software Engineering Terminology. Std. 610. 12-1990. Piscataway: IEEE Press, 1990.
7. IEEE. Recommended Practice for Software Requirements Specifications. Std. 830-1998. Piscataway: IEEE Press, 1998.
8. IEEE. Recommended Practice for Software Design Descriptions. Std. 1016-1998. Piscataway: IEEE Press, 1998.
9. IEEE. Standard for Software Verification and Validation. Std. 1012-1998. Piscataway: IEEE Press, 1998.

Note

1. This article uses the definitions from IEEE/EIA 12207.0-1997, where acquirer is used for customer and supplier is used for developer or contractor.

About the Author

Richard Hall Thayer, Ph.D., is a senior lecturer for Software Management Training, LLC, and is a professor emeritus in software engineering at California State University, Sacramento. He is a retired Air Force colonel and managed many of the Air Force's software engineering projects. Thayer is also a consultant in software engineering and project management and a visiting researcher and lecturer at the University of Strathclyde, Glasgow, Scotland. He has written more than 50 papers and books on software engineering, including two software engineering standards. Thayer received his doctorate in electrical engineering from the University of California at Santa Barbara and a master's and bachelor's in engineering degrees from the University of Illinois at Champaign/Urbana.

Software Management Training, LLC
6540 Chiquita WY
Carmichael, CA 95608
E-mail: r.thayer@computer.org

Coming Events

October 2-3
Department of Homeland Security Department of Defense Software Assurance Forum
Tysons Corner, VA
https://buildsecurityin.us-cert.gov/daisy/bsi/events.html

November 4-7
AYE 2007
Amplifying Your Effectiveness
Phoenix, AZ
www.ayeconference.com/conference.html

November 8-9
Static Analysis Summit II
Fairfax, VA
https://buildsecurityin.us-cert.gov/daisy/bsi/events.html

November 12-16
ICSPI 2007
Orlando, FL
www.icspi.com

November 14-16
The 10th IEEE High Assurance Systems Engineering Symposium
Dallas, TX
http://hase07.utdallas.edu

November 19-21
The 11th International Conference on Software Engineering and Applications
Cambridge, MA
www.iasted.org/conferences/cfp-591.html.

May 2008
Systems and Software Technology Conference
www.sstc-online.org

Coming Events: **Please submit coming events that are of interest to our readers at least 90 days before registration. E-mail announcements to: nicole.kentta@hill.af.mil.**

Issues Using DoDAF to Engineer Fault-Tolerant Systems of Systems

Dr. Ronald J. Leach
Howard University

It is unreasonable to expect all portions of complex, safety-critical, net-centric systems to always function properly. Hence, fault-tolerance techniques are necessary to ensure overall satisfactory system operation. This article considers some aspects of the relatively new Department of Defense Architectural Framework (DoDAF) in the systems engineering of complex systems that can be used to improve fault tolerance. It also describes some apparent deficiencies of DoDAF from a fault tolerance perspective and provides some suggestions for improvement.

There have been many advances in fault tolerance techniques since the early work of Brian Randell [1] on safe-state rollback and Algiradis Avizienis [2] on multiple code version redundancy. Several computer languages, including Ada, and operating systems have support for exception handling or fault isolation. Improved software engineering practice, especially the Software Engineering Institute's Capability Maturity Model® (CMM®) and CMM Integration℠ (CMMI®), has led to better data collection and analyses of faults and failures, thereby improving safety-critical systems [3, 4].

Unfortunately, there has been little research on the direct impact of fault tolerance techniques on systems engineering and requirements engineering of *systems of systems* (SoS), large systems that, themselves, are composed of multiple systems. The theme of this article is that there are opportunities to engineer fault tolerance into complex systems early in the life cycle, not just focusing on better ways to encode designs. The point is illustrated with a discussion of the use of DoDAF [5, 6, 7]. It is important to keep in mind that DoDAF is relatively new, with version 1.0 having been released in February 2004. Given the lead time for the development of complex systems, it is not reasonable to expect detailed analyses of DoDAF-based systems development, much less lessons learned. Some academic analyses have been performed; one example of such an analysis is [8].

Fault Tolerance, Reuse, and Commercial Off-The-Shelf (COTS)

It is well known that it is impossible to test fully all possible execution paths in large software systems. Therefore, most software is deployed with some number of software errors or faults, which may lead to system failures in the sense that a failure occurs when a system does not meet its specifications of behavior. This situation occurs regardless of whether the system is entirely created from new code or, as most systems are created today, with a heavy use of reusable components of a variety of sizes.

Since most large safety-critical systems are released and deployed with some (often unknown) number of faults, techniques that allow the effect of faults to be isolated, to be prevented from propagating from the subsystem in which they originate, or to have their effects minimized are often critical. It is especially important to mitigate any fault that may occur in COTS products whose internal structure and potential faults are often extremely difficult to determine.

> *"Fault tolerance is critical, and support for it is likely to become more important in the future."*

Some of the work of Jeffrey Voas [9, 10] has provided important advances in fault-tolerant systems – the systematic analysis and prediction of fault propagation across the boundary of subsystems, especially COTS products. This clearly illustrates that software fault tolerance is intimately connected with reuse and systems engineering. Of course, many developers of COTS products are not concerned (or deeply concerned) with fault tolerance. However, the deployers of such products within safety-critical systems must be.

Most large, complex, software systems developed in the last two decades have attempted to reuse existing software components as much as possible. Reuse has enormous potential for cost savings, but poses some issues for development of fault-tolerant systems because of unknown quality of some components, especially COTS products.

Wrappers, *bridgeware*, or *glueware* are often created to provide an interface between applications, components, data stores, or systems. Their interfaces and functionalities are easy to specify during the requirements gathering period, assuming the high-level interfaces of items that are to be *wrapped* are known. David Corman discussed technologies for wrapper generation in a 2001 CrossTalk article [11].

Unfortunately, wrappers may hide unknown defects within individual system components, since the internal interfaces and quality of COTS products are often difficult or impossible to determine.

The timing performance of systems with wrappers can be hard to determine. It can be difficult to determine if some actions are atomic (non-interruptible), which is often a requirement for system correctness; coordination of atomic actions across networks is even harder to guarantee. There are some fault-tolerance issues that are caused directly by improper use of wrappers which are discussed briefly when (Operational View) OV-6c diagrams are described later in this article.

Recovery blocks (areas of *safe* code) and redundancy are likely to be useful for functions, procedures, or objects that reside on a single system. Wrappers, bridgeware, or glueware are highly likely to have to function across two or more servers, depending on where the applications being *glued* reside. The applications themselves may require the use of several different servers. Hence, it can be difficult to find a safe state where all servers are consistent. Guaranteeing coordinated atomic actions is difficult, but essential, if fault tolerance can be achieved.

It is difficult to provide fault-tolerance for COTS products because generally

there is little knowledge of the internal interfaces of the product. Exception handling is usually done in wrappers and operating systems. Fault tolerance in operating systems can be helpful here if the COTS product is running on a single server. If the COTS product requires running on multiple servers, the situation becomes much more complex.

Systems created from COTS products often have predictable initial cost, although maintenance costs may be much larger than expected for long-lived systems, as Voas [10] and Leach [12] suggest. Additional problems occur if the schedule of releases of new versions of a COTS product is out of phase with the system to be deployed or the vendor of the COTS product stops product support during the desired system's life cycle.

The most extreme case of reusing software is new systems that are themselves composed of systems. These SoS will exhibit most of the difficulties indicated earlier, and to a much greater extent than what is typical of the *simpler* cases described previously. Systems too large to run on a single server need redundant hardware to provide continuity of operation if failure occurs; redundancy may also be required on a single-server system.

SoS will exhibit most of the difficulties indicated earlier, and, in many net-centric systems, it is impossible to allocate a set of hardware to preserve system state in the event of an unforeseen fault. Hence these systems, especially safety-critical ones, are greatest in need of systems development processes that support and encourage fault tolerance.

Systems Engineering With DoDAF

DoDAF was developed to help with the design of complex SoS. DoDAF is designed to work with several types of software development methodologies, notations, and Computer Assisted Software Engineering (CASE) tools. In this section, we discuss the gap between the results of dependability research and the incorporation of these results into the engineering of complex systems. Most dependability research focuses on lower-level actions such as exception handling, often with new languages [13]. Our DoDAF experience is with Telelogic's high-quality System Architect for DoDAF (Vers. 10.3.19), which supported all modeling requirements of a portion of a large multi-year project.

DoDAF supports the creation of several types of views, which provide insight into the complete requirements, design, and development process. The views can be classified into three broad categories: OVs, systems views (SVs), and technical standards views (TVs), each of which conveys different aspects of the architecture in several products. All views can be augmented by providing context, summary, or overview-level information or an integrated dictionary to define terms.

OVs depict what is going on in the real world that is to be supported or enabled by systems represented in the architecture. Activities performed as parts of system operations and the associated information exchanges among personnel or organizations are the primary items modeled in OV. The OV reveals requirements for capabilities and interoperability.

SVs describe existing and future systems and physical interconnections that support the needs of the originating funding organization that are documented in the OVs.

TVs are used to catalog standard system parts or components and their interconnections. This view augments the systems view with technical detail and forecasts of standard technology evolution. The terms *technical view* and *architectural view* are sometimes used instead, especially in documents before release 1.0 of the DoDAF handbook was widely available.

The potential for inclusion of fault tolerance in each type of DoDAF view is discussed in turn. For completeness, some descriptive material is included in this section with minor changes to meet space limitations; the material is taken from the public domain DoDAF materials provided by the DoD Architecture Working Group as listed in the references.

The first set of DoDAF views described here is known as OVs, which show the essence of system operations. These views are illustrated with diagrams describing the docking of the Crew Exploratory Vehicle (CEV) with the International Space Station (ISS) under the guidance of ground control.

High-Level Operational Concept (OV-1)

This is an informal, cartoon-like, graphical diagram intended for high-level presentations. Fault tolerance is not usually illustrated here. Our OV-1 example diagram (Figure 1) shows the tracking and relay satellite system interface of the ISS and CEV. The icons and connections are not entered into any internal data structures of the CASE tool and, thus, are not available to the rest of the representations.

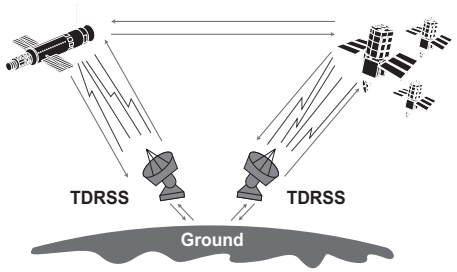


Figure 1: *High-Level Operational Concept (OV-1)*

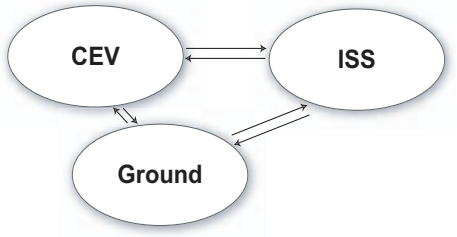


Figure 2: *Operational Nose Connectivity (OV-2)*

Operational Node Connectivity (OV-2)

Figure 2 tracks exchange of information from key operational nodes. Redundant connections to provide fault tolerance using backup systems and recovery routines can be specified here in detail using OV-2 *needlines*. The detailed information needed by the operational nodes is not displayed in this diagram in Telelogic's System Architect for DoDAF tool; such data is, however, stored internally in a database and is visible in the OV-3 matrix.

Operational Information Interchange (OV-3)

This matrix expresses relationships between the three basic OV architecture data elements: operational activities, operational nodes, and information flow. The matrix is generated automatically from the detailed information entered in the OV-2 database. This matrix has the same fault-tolerance aspects as OV-2 diagrams.

Organizational Relationships (OV-4)

This chart clarifies relationships between organizations and sub-organizations. It is not applicable to fault tolerance.

Operational Activity Model (OV-5)

Figure 3 (see page 24) delineates lines of responsibility for activities; uncovers redundancy; suggests decisions about streamlining, combining, or omitting activities; and defines or flags issues that need to be scrutinized further. It is the basis for OV-6 depictions of activity sequencing and timing. Fault tolerance can be incorporated here explicitly, since *operational control activity redundancy* (a DoDAF

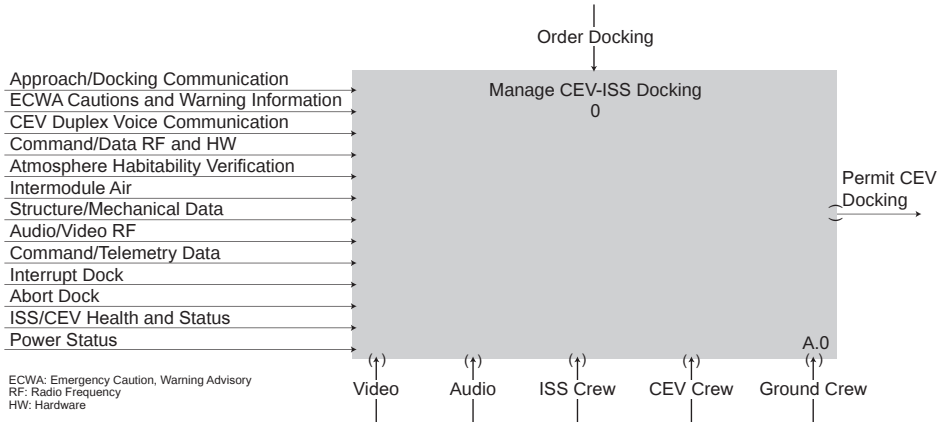


Figure 3: *Operational Activity Model (OV-5)*

term) is a primary example of fault tolerance.

An OV-5 diagram is shown for the parent operational node; there are similar ones (not shown) for each of the child nodes. The arrows are Input, Control, Output, and Mechanism (ICOM). An ICOM diagram always has the Input arrows on the left of an Operational Activity Node, Control arrows on the top, Output arrows on the right, and Mechanism arrows on the bottom. The mechanism arrow is an obvious place for representation of an alternate process if a fault is detected and a secondary communications channel (input or output) must be used. Any primary or secondary communications channels can be included in an OV-5 diagram.

Operational Activity Sequence and Timing Descriptions (OV-6)

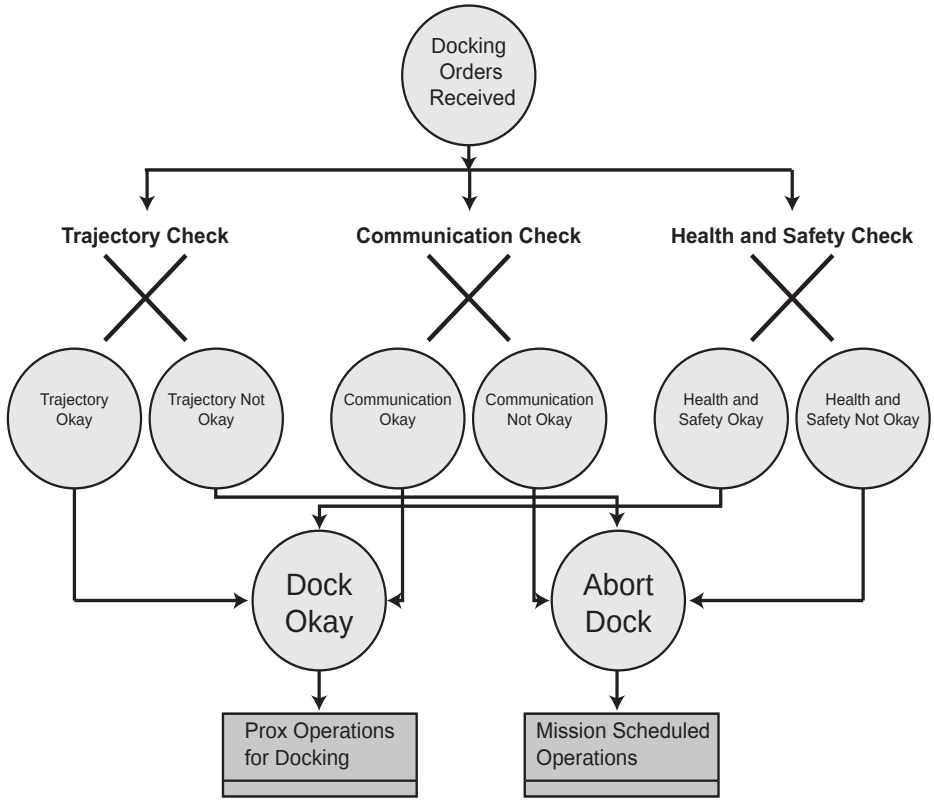
There are three types of OV-6 descriptions intended to show critical characteristics of dynamic sequencing and timing. Fault tolerance can be incorporated into each of them.

Operational Rules Model (OV-6a)

Figure 4 describes what a system does, using representations such as statecharts, Petri Nets, and process flow diagrams for the timing of processes and availability of operations. Statecharts are used in our example.

Fault tolerance, at least for known types of faults, can be represented here in detail. Icons listed as *trajectory not OK* and *communication not OK* represent a simple place to show the appropriate redundancy and fault isolation actions to ensure a fault-tolerant system operation.

Figure 4: *Operational Rules Model (OV-6a)*



Operational State Transition Diagram (OV-6b)

Figure 5 describes a system and its transitions from an object-centered view. A top-level OV-6b diagram is essentially a decision tree. Fault tolerance can be indicated in the form of messages when faults are detected. However, there seems to be little DoDAF support for tolerance of unanticipated faults (for which messages indicating faults may not be sent).

Operational Event-Trace Description (OV-6c)

Figure 6 (see page 26) can be represented as a set of timelines and interactions for each important physical object in the system with control capabilities. Fault tolerance, at least for known types of faults that can occur only within certain time constraints, can be represented here in detail.

Notice that in an OV-6c diagram, there are two kinds of events: those entirely on a single system and those that involve communication between systems. No assumptions can be made about what happens on a system's timeline until an external event linking the system to another occurs. Thus it is possible for the first, asynchronous, communication from the CEV to the ground system to be received *after* the second communication (the third arrow) from the CEV to the ISS is received by the ISS, regardless of it being sent from the CEV *before* the first asynchronous communication to the ground system. It is possible that an outside event can reach a system while the system is executing a wrapper, causing a fault and leaving the system in an inconsistent state. Examples of this are well known in operating systems; see for example [14]. Synchronous communications are represented using explicit acknowledgement.

The complete set of DoDAF *system views* is discussed next. These views describe systems and interconnections providing for, or supporting, organizational functions including both operational and business, as well as associate system resources to the OV. DoDAF system views are often required as part of the software development process, especially when developing complex systems. Particular emphasis is placed on those views where fault tolerance can be included most easily. No diagrams are provided for reasons of space; see the DoDAF Deskbook [5] for a more detailed description of these views.

Systems Interface Description (SV-1)

This identifies nodes, systems, and system items and their interconnections, mapping systems by their interfaces to the nodes and needlines described in OV-2. Any fault tolerance using backup systems and recovery routines is carried over directly from OV-2 diagrams. Systems responsible for fault tolerance, including backup systems can be specified here. However, no formal language has been provided for this specification.

Since there is no formal language or methodology to support fault tolerance directly within DoDAF, one workaround is to annotate these and all related diagrams. Unfortunately, the annotations will not be inserted directly into a CASE tool's information repository; hence, the CASE tool's internal consistency checking will not be available.

Systems Communications Description (SV-2)

This describes details of links in an SV-1 diagram into communications nodes, paths, and networks. Techniques to handle known classes of faults of specific nodes paths, or networks, can be incorporated into this diagram. Any hardware redundancy for fault tolerance, including standby systems, can be specified here.

Systems-Systems Matrix (SV-3)

This matrix shows relationships among systems in a given architecture such as system-type interfaces, planned versus existing interfaces, etc. In theory, it is useful in identifying alternate forms of communication between systems if there is a fault. However, there is no formal way of specifying such alternate communication within DoDAF. One potential workaround is to manually include the appropriate alternative relationships. As before, such annotations are not generally included within the CASE tool's internal representation of the systems within the architecture and, thus, the tool's consistency checks are not done automatically.

Systems Functionality Description (SV-4)

This describes functions performed by systems and the system data flows between system functions. Lower-level diagrams can be either functional or data-flow decompositions of top-level diagrams. Since faults theoretically can occur whenever data is created, processed, or stored, fault tolerance procedures can be incorporated directly into these diagrams and, therefore, into the CASE tool's internal repository for system consistency analysis.

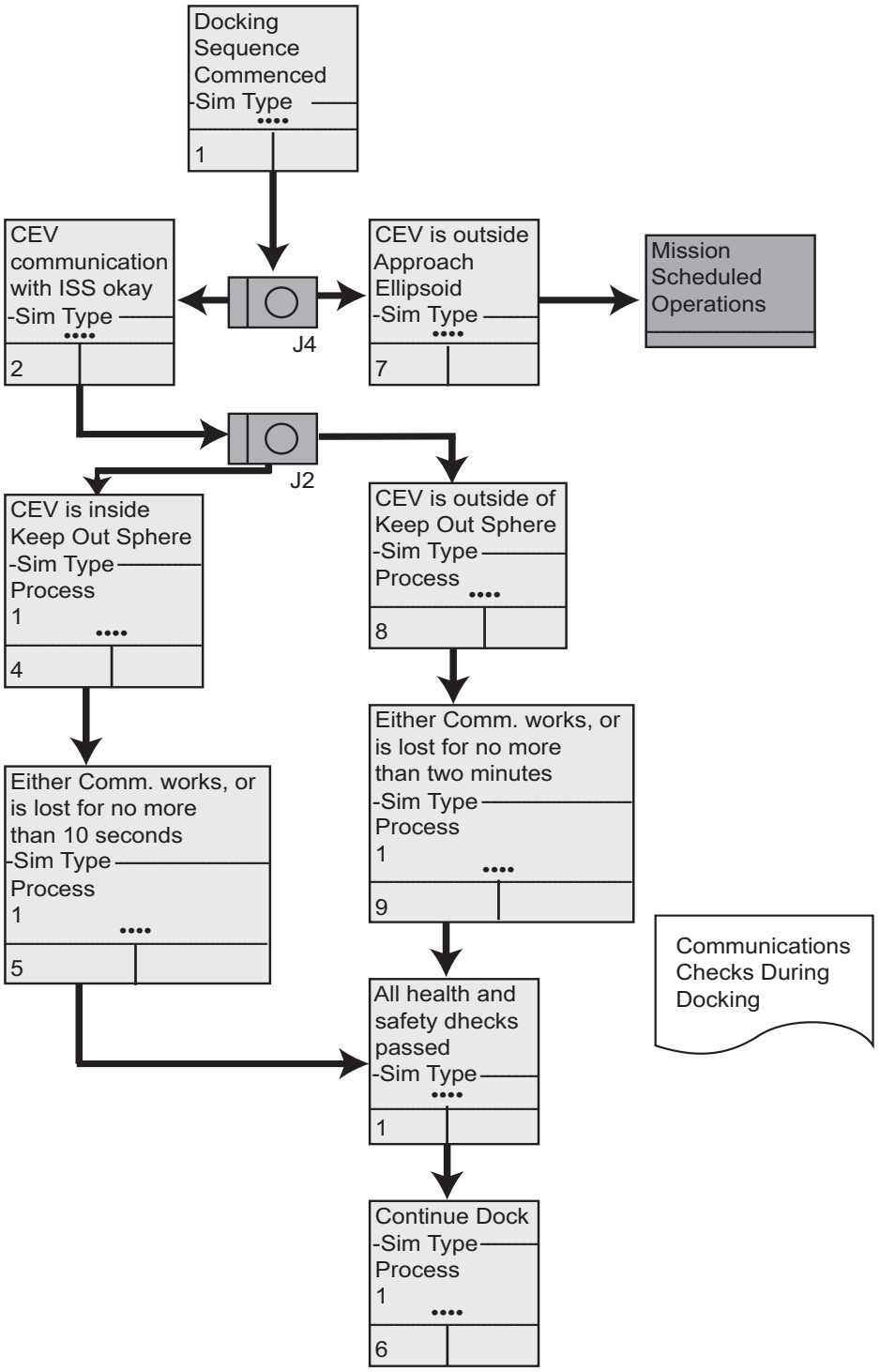


Figure 5: *Operational State Transition Diagram (OV-6b)*

Operational Activity to Systems Function Traceability Matrix (SV-5)

This provides mappings either of systems to capabilities or of system functions to operational activities. It is important to know which systems may be affected by a fault in an operational node; hence, fault tolerance is important here. Alternate safe states, recovery blocks, and redundant hardware can be specified for anticipated faults. Unfortunately, there is no obvious way to indicate alternative software/hardware paths in the event of an unknown failure.

Systems Data Exchange Matrix (SV-6)

This provides details of system data elements being exchanged between systems and their attributes. Since faults theoretically can occur whenever data is created, processed, or stored, fault tolerance can be incorporated anywhere in this matrix (as was the case with SV-4 diagrams).

Systems Performance Parameters Matrix (SV-7)

This provides performance characteristics

of SV elements for appropriate time frames. System faults due to distributed denial-of-service attacks can be addressed here. Delay-tolerant networks can serve as a useful recovery system. (An introduction to delay-tolerant networks is provided in [15].)

Systems Evolution Description (SV-8)

This lists planned incremental steps toward migrating a suite of systems to a more efficient suite or toward evolving a current system to a future implementation. This is the primary place where assessment of COTS products and their interfaces come into play when considering future fault tolerance.

Systems Technology Forecast (SV-9)

This is a text-based prediction of emerging technologies and software/hardware products, including assessment of COTS products and their interfaces, that are expected to be available in a given time period and that will affect future development of the architecture.

Systems Rules Model (SV-10a)

This identifies constraints that are imposed on system functionality due to some aspect of systems design or implementation. Such rules are often written in a relatively formal version of English (often called *Structured English*) using IF-THEN rules to indicate, say, timing or performance requirements. A more formal description than this would be useful for improving fault tolerance.

Systems State Transition Description (SV-10b)

This identifies responses of a system to events. The diagram is a systems-level version of OV-6b diagrams. This is clearly a place for fault tolerance, especially since linkages and nodes are placed directly into the CASE tool's internal repository for system consistency analysis.

Systems Event Trace Description (SV-10c)

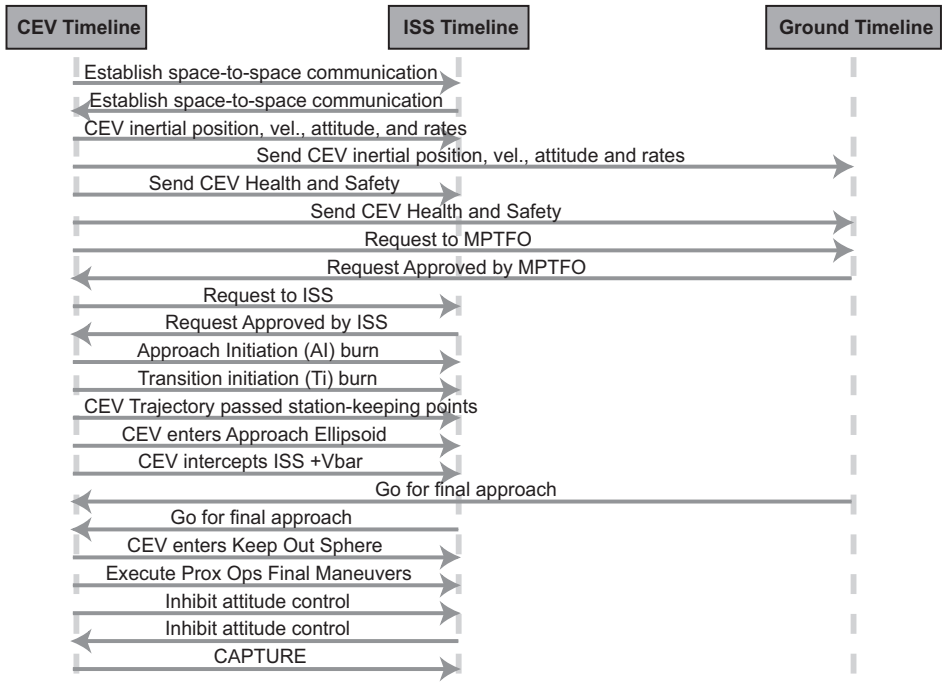
This is also used to describe system functionality. It identifies system-specific refinements of critical sequences of events described in OVs. The diagram is a systems-level version of OV-6c diagrams. This is clearly a place for fault tolerance especially since linkages and nodes are placed directly into the CASE tool's internal repository for system consistency analysis.

Physical Schema (SV-11)

This describes the physical implementation of the Logical Data Model entities to physical schema. Since this is the primary place where the failure of a server or data store is described in DoDAF views, hardware redundancy should be used, at least, here to improve system fault-tolerance.

There are two other types of DoDAF views that are not discussed here because they do not directly concern fault-tolerant systems: TVs, which list current and forecast standards, and *all views*, which include any overarching aspects, including doctrine; tactics, techniques, and procedures; goals and vision statements; concepts of operations; scenarios; and environmental conditions.

Figure 6: *Operational Event Trace Description (OV-6c)*



MPTFO: Mission Planning Training and Flight Operation

Conclusion and Further Work

It is clear that DoDAF is a complex framework that was intended for an even more complex problem – creating an SoS. The process for designing even a portion of a moderate sized system using DoDAF and a typical CASE tool is time-consuming, requiring several online tutorials and multiple presentations, along with hands-on help. One would expect strong support for fault tolerance, especially since DoDAF was intended initially for military systems. Indeed, the timing requirements that could be represented easily, say, in the OV-6a, OV-6b, and OV-6c operational views, and SV-10a, SV-10b, and SV-10c system views might include fault tolerance.

However, the reality is otherwise. The three most readily available sources of DoDAF documentation – the DoDAF deskbook, and volumes I and II of the DoDAF architectural framework – do not even mention the word fault. This is a strong indication of the lack of support for fault tolerance in the use of DoDAF within a systems engineering process. Again, this suggests an opportunity for having fault tolerance research incorporated into large-scale systems engineering methodologies and into high-quality industrial CASE tools.

There are many opportunities for the dependability community to increase research collaborations, especially in the engineering of SoS. The development of specific graphical notations for fault tolerance, such as coordinated atomic actions, and their eventual incorporation into commercial CASE tools and large architectural frameworks provides the potential to have fault tolerance built into systems rather than being an add-on.

Development of an expert system module to be incorporated into CASE tools holds considerable promise. Fault tolerance is critical, and support for it is likely to become more important in the future. Perhaps this support can be in the form of an expert system add-on to advise on fault tolerance, or in creation of a graphical notation that explicitly supports fault-tolerant designs and implementations. Having such support early in the life cycle can have a great effect on the creation of real fault-tolerant systems.

The convergence of the computer security and dependability fields is highly encouraging, since it provides an opportunity to study fault isolation and mitigation techniques during denial-of-service and

similar attacks.

A major goal of this article is to provide an impetus for the dependability community to affect the functionality of CASE tools and frameworks to provide early life-cycle support for fault tolerance. Clearly, the lack of early life-cycle support for fault tolerance makes it more difficult to include it within the design of complex SoS. It remains to be determined if there will be enough perceived return on investment in order for CASE tool vendors to add fault tolerance to DoDAF.◆

Acknowledgement

This research was partially supported by the National Science Foundation under grant number 0324818. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either express or implied, of the U. S. Government.

References

1. Randell, B. "System Structure for Software Fault Tolerance." IEEE Trans. Software Engineering 11.2 (1975).
2. Avizienis, A., and J.P. Kell. "Fault Tolerance by Design Diversity: Concepts and Experiments." Computer Aug. 1994: 67-80.
3. CMMI Product Team. "CMMI for Development." Version 1.2. Technical Report CMU/SEI-2006-TR-008. Pittsburgh: Software Engineering Institute (SEI), Carnegie Mellon University (CMU), 2006.
4. Capability Maturity Model. Version 1.0. Pittsburgh: SEI/CMU, 1991.
5. DoD. DoDAF v1 Deskbook. DoD Architecture Working Group, 2004 <https://acc.dau.mil/CommunityBrowser.aspx?id=31667>.
6. DoD. DoDAF v1 Volume I. DoD Architecture Working Group, 2004 <jitc.fhu.disa.mil/jitc_dri/pdfs/dodaf_v1v1.pdf>.
7. DoD. DoDAF v1 Volume II. DoD Architecture Working Group, 2004 <jitc.fhu.disa.mil/jitc_dri/pdfs/dodaf_v1v2.pdf>.
8. Mittal, Saurabh. "Extending DoDAF to Allow Integrated DEVS-based Modeling and Simulation." JDMS: The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology 3.2 (2006).
9. Voas, Jeffrey M. "COTS and High Assurance: An Oxymoron?" Proc. of the 4th IEEE International Symposium on High-Assurance Systems Engineering (HASE '99), November 17-19 1999, Washington: IEEE Computer Society, 1999.
10. Voas, Jeffrey M., and Jeffrey E. Payne. "Dependability Certification of Software Components." Journal of Systems and Software 52.2-3 (2000): 165-172.
11. Corman, David. "The IULS Approach to Software Wrapper Technology for Upgrading Legacy Systems." CrossTalk Dec. 2001 <www.stsc.hill.af.mil./crosstalk/2001/1201>.
12. Leach, Ronald J. "Can this COTS-Based System Be Saved?" PC/104 Embedded Solutions 9.4 (2005): 38-44.
13. Oliveira Guimaraes, Jose. "The Green Language Exception System." The Computer Journal 47.6 (2004): 651-661.
14. Silberschatz, A., Galvin, P., and G. Gagne. Operating Systems Concepts. New York: John Wiley, 2005.
15. Li, Jiang, et al. "Customizable Localized Computation of Connected Dominating Sets for Self-Organizing Wireless Networks." Proc. of the Second International Conference on Embedded Software and Systems (ICESS'05). IEEE, 2005.

About the Author

Ronald J. Leach, Ph.D., is professor and chair of the Department of Systems and Computer Science at Howard University. He does research in software engineering with special interest in reuse, metrics, fault tolerance, performance modeling, process improvement, and the efficient development of complex software systems. Leach is the author of five books and more than 65 other published technical articles. He has bachelor's, master's, and doctorate degrees in mathematics from the University of Maryland, and a master's degree in computer science from Johns Hopkins University. He has three children, a terrific grandson, two grandcats, and one granddog.

Department of Systems and Computer Science
School of Engineering
Howard University
Washington, D.C. 20059
Phone: (202) 806-6650
Fax: (202) 806-4531
E-mail: rjl@scs.howard.edu

A Framework for Evolving System of Systems Engineering

Dr. Ricardo Valerdi, Dr. Adam M. Ross, and Dr. Donna H. Rhodes
Massachusetts Institute of Technology

We provide a framework for examining the differences between systems engineering and system of systems engineering (SoSE). By taking normative, descriptive, and prescriptive views of these constructs, similarities and differences can be better identified. Moreover, we note that additional work is needed in the development of normative and prescriptive models in order to advance our understanding of both systems engineering and SoSE.

There is an ongoing debate that questions whether there are differences between a system and a system of systems (SoS). For purposes of this discussion, we define a system to be *a construct or collection of different elements that together produce results not obtainable by the elements alone.* The elements – or parts – can include people, hardware, software, facilities, policies, and documents; that is, all things required to produce systems-level results [1]. On the other hand, an SoS is a condition where a majority of the following five characteristics are present: operational independence, managerial independence, geographic distribution, emergent behavior, and evolutionary development [2]. There are numerous definitions for both systems and SoS, but in essence there are three schools of thought that exist on the issue of similarities and differences.

The first school believes that there are fundamental differences between the systems and SoS and, as a result, they warrant different names, methodologies, and tools to bring them to realization. The *pro-difference* camp appears to represent a majority on the debate, as evidenced by the amount of its advocacy across government, industry, and academia. Examples within the last five years include the following:

- Inauguration of the Institute of Electrical and Electronics Engineers (IEEE) Conference on SoS.
- Inception of the International Journal of SoS Engineering.
- Definition of the SoS signature area at Purdue University.
- Creation of the National Center for Systems of Systems Engineering at Old Dominion University.
- Inclusion of SoS considerations in the Systems Engineering Chapter of the Defense Acquisition Guidebook [3].
- Procurement and development of systems uniquely labeled as System-of-Systems such as the Army's Future Combat Systems by Boeing, Science Applications International Corporation, and thousands of subcontractors.
- Creation of the SoS Engineering Center of Excellence by the Office of the Under Secretary of Defense for Acquisition, Technology and Logistics, specifically the Deputy Director of Joint Force Integration.

The second school believes that SoS is simply an unnecessary term for a system, and that society should not be influenced by this subtlety. To follow Rechtin's heuristic: One man's architecture is another man's detail [4]. Similarly, an SoS for one organization may be seen as a system to another. The advocates for this approach believe that traditional systems engineering practices along with effective program management will be sufficient for both systems and SoS.

The third school is agnostic on the issue, playing the *wait-and-see* game. Since there may not be enough evidence to prove that there is or is not a difference, then there is no sense in claiming loyalty to either of the first two schools of thought to avoid the risk of being wrong. This opinion represents the minority, since from a practical perspective there are few engineers who are unwilling to take a side on technical issues.

At the center of the debate about the semantic difference between systems and SoS are the engineering activities that are involved in each. For the definition of systems engineering, for example, we can turn to widely accepted standards [5, 6, 7] used to define the what and how of the activities needed to engineer a system. However, consensus has not been reached on the activities needed to engineer an SoS. Despite the ongoing debate, organizations need to take action on the systems they want to create. But how does one make sense of these concepts? The following section provides a framework to help make sense of this debate.

Normative, Descriptive, and Prescriptive Framework

To help structure the discussion, a framework is proposed that provides a categorization of concepts into normative, descriptive, and prescriptive models. For purposes of this discussion, the term model is used to include formal and informal representations of structure.

A normative model is one that represents norms or cultural standards. Similarly, a normative statement describes how the world *should* be; it provides a yardstick to measure whether something is good. One example is the IEEE 1220 standard entitled *Application and Management of the Systems Engineering Process* [5]. As a normative model, it describes how systems engineering ought to be implemented for the realization of successful programs. Because these standards result from reflection and analysis, normative models are representative of the type of models developed in philosophy, mathematics, and other theory-based disciplines [8]. In the domain of systems, the development of normative models relies on assumptions about how systems will be implemented, and through interpreting the implementation, suggests the best way to build systems. Normative models are analogous to first principles of nature since they are foundational propositions from which all other propositions can be deduced.

A descriptive model characterizes actual behavior of decision-makers, or how the world *actually is.* An example is the set of systems engineering case studies developed by the Center for Systems Engineering at the Air Force Institute of Technology [9]. These include detailed accounts of how systems engineering was actually done on large programs such as the B-2 stealth bomber, Theater Battle Management Core System, and the Joint Air-to-Surface Standoff Missile. As descriptive models, these case studies are a representation of our understanding of how systems engineering has been done in the past, whether good or bad. Deviations from a normative model (i.e., an IEEE standard) can be highlighted through a descriptive model. Descriptive models capture not only behavior, but also decision-making, insofar as the outcomes of the decisions are discernable, or elicited from decision-makers. A classical example of a descriptive model is a regression analysis on empirical data. The *best fit* curve to a set of data captures the essence of empirical reality and can be used to predict the future but is limited by the context and conditions of the original data underlying the model.

Descriptive models also provide insight into organizational practices of systems engineering. Some organizations, for example, may think that it is appropriate to allocate systems engineering functions (i.e., testing, documentation) to software engineers while others may decide to treat those as completely separate functions. A descriptive model would relate these differences without a justification in terms of a normative *right* or *wrong* basis, but rather just in terms of stating the specific process used in organizations.

A prescriptive model is one that aims at correcting biases with the intent to improve judgments and decisions according to normative standards. In other words, a prescriptive model is based on advice on how to best achieve the ideals suggested by the normative view, given the facts highlighted through the descriptive view. An example of a prescriptive model is one that provides practical directions on how to implement process improvement through Capability Maturity Model Integration (CMMI®) in organizations that develop systems or SoS. It contains the prescriptive representation of how it *can* be done and a strategy on how to get from the present state (descriptive) to ideal state (normative) as viewed by best practices in the software defense industry.

Equipped with these concepts, the differences between systems and SoS can be considered through a more analytical perspective. First, a descriptive view is provided, followed by a discussion on how normative and prescriptive models can be developed to further compare systems and systems of systems.

A Descriptive View of Systems

Clearly, there are similarities between systems and SoS. Some of these similarities are the following:

- Both can be very complex.
- Both involve people, hardware, software, facilities, policies, processes, etc.
- Both have purpose.

Even though both systems and SoS share many traits, it is the difference between them that provides the basis for debate. A helpful descriptive view of the issue is to compare what is involved with engineering a system to engineering an SoS. Academia [10] and government [11] have highlighted key differences between the two. We provide an adaptation of previous work [12] that highlights salient descriptive differences between traditional systems engineering practice and SoSE practice (see Table 1).

Note that not all systems will contain the attributes in the middle column and not all SoS will contain the attributes of the third column, but in general a SoS always requires at least some of the elements of SoSE.

Presently, there are a number of descriptive research efforts that are helping inform the debate about the differences between system and SoS. In the area of cost estimation, work is being done to define the unique attributes of SoS for purposes of developing a cost model to estimate the SoS integration effort [13, 14]. This work was motivated by the inability of existing system-level cost models to estimate the effort needed to integrate SoS.

	Traditional Systems Engineering	SoSE
Purpose	Development of a single system to meet stakeholder requirements and defined performance.	Evolving new system of systems capability by leveraging synergies of legacy systems and emerging capabilities.
Systems Architecture	Established early in the life cycle; expectation set remains relatively stable.	Dynamic adaptation as emergent needs change.
System Interoperability	Interface requirements are defined and implemented for the integration of components in the system.	Component systems can operate independently of SoS in a useful manner; protocols and standards are essential to enable interoperable systems.
System *ilities*	Reliability, maintainability, and availability are typical concerns.	Enhanced emphasis on *ilities* such as flexibility, adaptability, and composability.
Acquisition and Management	Centralized acquisition and management of the system.	Component systems separately acquired and continue to be managed and operated as independent systems.
Anticipation of Needs	Concept phase activity to determine system needs.	Intense concept phase analysis followed by continuous anticipation, aided by ongoing experimentation.
Cost	Single or homogenous stakeholder group with stable cost/funding profile and similar measures of success.	Multiple heterogeneous stakeholder groups with unstable cost/funding profile and measures of success.

Table 1: *Areas of Emphasis in Systems Engineering and SoSE [12]*

As a first step toward a prescriptive model, the Department of Defense (DoD) is developing a Guide to System of Systems Engineering [11], based on best present state knowledge. The guide provides 16 DoD technical and management processes to help sponsors, program managers, and chief engineers address the unique considerations for DoD SoS. Results from these efforts will serve to further characterize the state of the practice of systems engineering and may begin to identify enablers, barriers, and successful techniques for SoS engineering practice. Ultimately, the observation of real systems and SoS enable the development of prescriptive models that are grounded in reality and eventually generate useful prescriptive guidance. It is important to recognize that implicit in any prescriptive model is a normative basis for why a particular guidance is suggested. Users of such guidance should make an effort to recognize that basis to better understand the rationale for the guidance and under what conditions that guidance may be ill-advised.

Ideally, a normative model is developed prior to any prescriptive model. In a practical sense, these often evolve a bit differently. For example, the DoD Guide to System of Systems Engineering implicitly relies upon normative models for traditional systems engineering [15], due to a lack of SoS normative models [16]. In the future, normative models for SoS are needed in order to provide a sound and defensible basis for *good* SoSE practice. Researchers and practitioners alike should strive towards a better understanding of successful systems and SoS and the development of normative models to improve understanding of the fundamental differences between successful systems and SoS. Recent efforts to define a research agenda for SoS architecting [17] demonstrate the need for government, academia, and industry to work together to advance the state of affairs.

Practical Implications

The immaturity of normative models as discussed has several implications for acquirers and implementers of SoS. First, there are no industry best practices that can be used as normative models to compare how SoS acquisitions and implementations should be done. Such a model would describe the rationale for how *best* is defined and how it could be accomplished in an ideal scenario. A natural progression suggests that descriptive

models will come first through the observation of successful and unsuccessful SoS. Subsequently, understanding how SoS should be acquired and implemented are derived into normative models. Once a descriptive model of the present and a normative model of the *ideal* future are developed, a thorough benchmarking process can begin. The development of practical strategies and processes, such as those expressed in a guide, require the codification of prescriptive models. Since prescriptive models require descriptive and normative models as their basis, the better the descriptive and normative models, the better the prescriptive advice.

A second implication is that there is a lack of reference costs and schedule estimates for SoS. This is a crucial void for predictive estimation models since they require historical data for their development and calibration. This also emphasizes the need to capture lessons learned from completed programs to develop descriptive models.

Returning to the debate on how the engineering of a system differs from that of a SoS requires a sound theoretical and practical foundation on what these two processes entail. Much of the research to date on SoS involves descriptive type approaches. Advancing prescriptive guidance on how to do SoSE requires a normative model as well. The absence of validated normative models presents an opportunity for industry, government, and academia to collaborate. It is only when that piece falls into place that a rigorous and defensible development of a standardized and successful SoSE process can take place.◆

References

1. Rechtin, E. The Art of Systems Architecting. New York: CRC Press, 2000.
2. Sage, A., and C. Cuppan. "On the Systems Engineering and Management of Systems of Systems and Federations of Systems." Information, Knowledge, and Systems Management 2 (2001): 325-345.
3. Defense Acquisition University (DAU). Defense Acquisition Guidebook. Vers. 1.06. DAU, 2006.
4. Rechtin, E. Systems Architecting: Creating and Building Complex Systems. New Jersey: Prentice Hall, 1990.
5. "Application and Management of the Systems Engineering Process." IEEE 1220, 1998.
6. American National Standards Institute /Electronic Industry Alliance (ANSI/ EIA). "Processes for Engineering a System." ANSI/EIA-632-1988. ANSI/ EIA, 1999.
7. International Standards Organization/ International Electrotechnical Commission (ISO/IEC). "Systems Engineering – System Life Cycle Processes." ISO/IEC 15288:2002(E). ISO/IEC, 2002.
8. Baron, J. Thinking and Deciding. 3rd ed. Cambridge University Press, 2000.
9. Systems Engineering Case Studies. Air Force Center for Systems Engineering <www.afit.edu/cse/cases.cfm>.
10. Keating, C., et al. "System of Systems Engineering." Engineering Management Journal 15.3 (2003).
11. Office of the Under Secretary of Defense. System of Systems Systems Engineering Guide: Considerations for Systems Engineering in a System of Systems Environment. Vers. 0.9. DoD, 2006 <www.acq.osd.mil/se/publications.htm>.
12. Rhodes, D. "Evolving Systems Engineering for Innovative Product and Systems Development." Proc. From Massachusetts Institute of Technology (MIT) Systems Design and Management Alumni Conference, Oct. 2004.
13. Lane, J., and R. Valerdi. "Synthesizing SoS Concepts for Use in Cost Estimation." Systems Engineering 10.4 (2007).
14. Lane, J. "System of Systems Lead System Integrators: Where Do They Spend Their Time and What Makes Them More/Less Efficient?" USC-CSE-2005-508. Los Angeles: USC Center for Systems and Software Engineering, June 2005.
15. Sage, A.P. Systems Engineering. Wiley-Interscience, 1992.
16. U.S. Air Force. "Report on System of Systems Engineering for Air Force Capability Development." SAB-TR-05-04. July, 2005.
17. Axelband, E., et al. "A Research Agenda for Systems of Systems Architecting." Proc. of the 17th INCOSE Symposium, June 2007, San Diego, CA.

About the Authors

Ricardo Valerdi, Ph.D., is a research associate in the Systems Engineering Advancement Research Initiative (SEAri) and a lecturer in the Engineering Systems Division at MIT. His research interests are in systems engineering cost estimation and the synergies between systems engineering and software engineering. Valerdi has a doctorate in industrial and systems engineering from USC.

MIT
77 Massachusetts AVE
41-205
Cambridge, MA 02139
Phone: (617) 253-8583
Fax: (617) 258-7845
E-mail: rvalerdi@mit.edu

Adam M. Ross, Ph.D., is a research scientist in the SEAri at MIT. His research interests are in the exploration of system architecture tradespaces, and the advancement of dynamic value considerations through quantification and visualization during the system design process. Ross has a doctorate in engineering systems from MIT.

MIT
77 Massachusetts AVE
NE20-388
Cambridge, MA 02139
Phone: (617) 324-0473
Fax: (617) 258-7845
E-mail: adamross@mit.edu

Donna H. Rhodes, Ph.D., is the director of SEAri, Principal Researcher, and is a senior lecturer in the Engineering Systems Division at MIT. Her research interests are in advanced systems engineering methods and practices. Rhodes has a doctorate in systems science from the State University of New York, Binghamton.

MIT
77 Massachusetts AVE
NE20-388
Cambridge, MA 02139
Phone: (617) 324-0473
Fax: (617) 258-7845
E-mail: rhodes@mit.edu

Softwareitaville

Across the nation, students of all ages are begrudgingly penning summer vacation essays for English teachers. If your son or daughter is struggling, maybe they can borrow mine. I visited my son who spent his summer as a medical officer at a scout camp on Catalina Island.

Catalina veterans, who have traveled the entire island, know this is not your typical planes, trains, and automobiles excursion. It's more a planes, wait, taxis, wait, ferries, wait, safari buses, wait, golf-carts-and-hiking-boots type of escapade. Due to the island's small airport, rental car dearth, and plethora of Conservancy (read island mafia) restrictions, travel throughout the island relies on indigenous services and ... locals.

Island coterie restrictions are ecological, economical, and, at times, sadistic. Coming from the Mountain West, I'm accustomed to independent travel, coming and going as I please. After numerous calls to island proprietors, it was clear that independent travel – beyond walking – was out. Rather than buck the system, the engineer in me decided homework, planning, and execution would result in smooth travel. What I did not count on was island time.

Island time is an attitude, a state of mind that puts time on the bottom rung of the priority ladder. On island time, a restaurant advertising breakfast at 8:00 a.m. may open any time between 8:30 and 9:00 a.m. On island time, the 12:00 a.m. noise ordinance may be enforced around 1:00 or 2:00 a.m. On island time, the Safari Bus requiring passengers to be in line 30 minutes in advance may depart 30 to 40 minutes after a 10-minute delay because ticket holders outnumber bus seats. After 15 minutes on a dusty, bumpy road through the heart of the isle, the bus driver, under the influence of island time, stops to check if the back door of the storage compartment containing your luggage is closed.

On island time, there are two days of the week: today and tomorrow. Yesterday is a memory, anything beyond tomorrow is unfathomable, and anything that does not get done today can wait for tomorrow.

Don't get me wrong – island time can be relaxing and revitalizing. However, if you need to travel the length of the island, pick up your son, take him back the length of the island for a hot shower, steak dinner, and soft bed and then return the length of the isle, it can be frustrating. A kind of frustration one can suffer on software projects.

Island time is not much different than a project's inaugural time. Inaugural time occurs at the beginning of a project when requirements are few, resources abundant, and budgets profuse. On inaugural time, the system is perfect (in your mind), the customer is your friend (in your mind), and modules not completed today can wait until tomorrow (also in your mind).

Unlike island time, extended inaugural time is lethal. Left uncurbed, a refreshing project expedition turns into a tedious death march. Before you know it, the blame game commences and you are singing the blues. Fortunately, you can sing the following song at the company's project cancellation party at your local karaoke bar.

(Sung to the tune of Jimmy Buffet's *Margaritaville* ... my apologies, Jimmy):

Feelin' my wrist ache
Watchin' my drive bake
Management purists, pushing snake oil
Strummin' eighty-eight keys,
Trusting they won't freeze
Top brass wimps, they're beginnin' to roil

Wasted away again in Softwaritaville
Searchin' for my lost taker, of fault
Some people claim that there's a manager to blame
But I know it's nobody's fault

I don't know the reason
For customer treason
Nothing to show but this whip through code glue
But it's a real beauty
A pension plan booty, how it got here
I haven't a clue

Wasted away again in Softwaritaville
Searchin' for my lost taker, of fault
Some people claim that there's a client to blame
Now I think, it could be my fault

I blew out a flip flop
Stepped up an amp drop
Cut a deal; online buy from Saigon
But there's ruse in the vender
and soon I will tender
A lax obligation that helps us plough on

Wasted away again in Softwaritaville
Searchin' for my lost taker, of fault
Some people claim that there's a vendor to blame
But I know, it's my own darn fault

Some people claim that there's a hacker to blame
And I know it's my own darn fault

Promptly add the following to your process asset library: "All members of a failed project, as their last act, are to sing the failed-project anthem – Softwaritaville – at the project cancellation party." Enjoy.

— Gary A. Petersen
Arrowpoint Solutions
gpetersen@arrowpoint.us

CrossTalk / 517 SMXS/MXDEA
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

PRSRT STD
U.S. POSTAGE PAID
Albuquerque, NM
Permit 737

CrossTalk is co-sponsored by the following organizations:

NAVAIR

Homeland Security